МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ (Савченко А.С.)
«_____»_____2021 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "БАКАЛАВР"

Тема: "Застосунок відображення статистичних даних Формули-1"

Виконавець: студент УС-412 Садихов Едгар Халідович
(студент, група, прізвище, ім'я, по батькові) (підпис)

Керівник: к.т.н., доцент Климова Асія Сабирівна
(науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Нормоконтролер: ст. викл. Шевченко О. П.
(П.І.Б.) (підпис)

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної програмної інженерії

Кафедра комп’ютерних інформаційних технологій

Освітньо-кваліфікаційний рівень: Бакалавр

Галузь знань, спеціальність, спеціалізація: 12 “Інформаційні технології”, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Савченко А.С.
“_____”_____2021р.

ЗАВДАННЯ
на виконання дипломного проекту студента

Садихова Едгара Халідовича
(прізвище, ім’я, по батькові)

1. Тема проекту (роботу): «Застосунок відображення статистичних даних Формули-1» затверджена наказом ректора № 636/ст. від 22.04.2021р.
2. Термін виконання проекту (роботи): з 10.05.2021 до 20.06.2021р.
3. Вихідні данні до проекту (роботи): база даних статистики усіх чемпіонатів Формули-1, мова програмування серверної частини - Java, середовище програмування - IntelliJ IDEA 2021.1 Ultimate Edition.
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): аналіз статистичної інформації чемпіонату Формули-1, розробка модуля вилучення інформації з бази даних, розробка інтерфейсу користувача.
5. Перелік обов’язкового графічного матеріалу: "Технології та інструменти", "Структура бази даних", "Локальний сервер", "Графічний інтерфейс", "Табличне представлення даних", "Діаграма класів системи".

КАЛЕНДАРНИЙ ПЛАН

№ п/п	*Етапи виконання дипломного проекту*	*Термін виконання етапів*	*Примітка*
1	Ознайомлення з предметною областю.	10.05.2021	
2	Складання змісту пояснювальної записки дипломного проекту.	11.05.2021 – 12.05.2021	
3	Аналітичний огляд і постановка задачі.	13.05.2021 – 14.05.2021	
4	Встановлення та оновлення усіх необхідних програмних засобів та інструментів.	17.05.2021 – 20.05.2021	
5	Аналіз статистичної інформації чемпіонату Формули-1.	21.05.2021 – 24.05.2021	
6	Робота з даними за допомогою системи керування базами даних MySQL.	25.05.2021 – 28.05.2021	
7	Розробка серверної частини застосунку.	31.05.2021– 02.06.2021	
8	Підключення бази даних до основного програмного модуля.	03.06.2021	
9	Розробка запитів до бази даних для вилучення необхідної користувачеві інформації.	04.06.2021– 05.06.2021	
10	Розробка графічного інтерфейсу застосунку.	06.06.2021 – 08.06.2021	
11	Робота над розділами пояснювальної записки.	08.06.2021– 10.06.2021	
12	Оформлення дипломної роботи та додаткового графічного матеріалу.	10.06.2021 – 11.06.2021	

Студент (Садихов Е.Х.)

Керівник дипломного проекту (Климова А. С.)

РЕФЕРАТ

Дипломний проект містить 58 сторінок, 1 таблицю, 11 рисунків, 11 літературних джерел.

Тема дипломного проекту «Застосунок відображення статистичних даних Формули-1».

Мета дипломного проекту полягає у розробці застосунку для відображення детальних статистичних даних за всю історію чемпіонату.

Об'єктом дослідження є інформаційні процеси системи групування та відображення статистичних даних.

Предметом дослідження є інформаційна система відображення статистики в контексті технології вилучення інформації з бази даних на персональному комп'ютері.

Відповідно до мети дослідження були поставлені та розв'язані наступні завдання:

- аналіз статистичних даних Формули-1;
- реалізація бази даних;
- розробка модуля вилучення інформації з бази даних;
- підключення бази даних до основного модуля;
- реалізація інтерфейсу користувача;

За результатами дослідження сформульовані та визначені послідовні операції для реалізації програми, щоб отримати статистичні дані за певними параметрами. Одержані результати можуть бути використані у діяльності спортивних журналістських видань, веб-ресурсів, де розглядаються етапи чемпіонату Формули-1.

БАЗА ДАНИХ; ІНТЕРФЕЙС; ЛОКАЛЬНИЙ ВЕБ-СЕРВЕР; ФРЕЙМВОРК; СТАТИСТИКА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Болід	–	автомобіль з відкритими колесами, створений спеціально для перегонів
FIA (Fédération Internationale de l'Automobile)	–	Міжнародна Автомобільна Федерація
ПЗ	–	програмне забезпечення
HTML (Hyper Text Markup Language)	–	мова розмітки гіпертексту
CSS (Cascading Style Sheets)	–	формальна мова опису зовнішнього вигляду документа
COVID-19	–	інфекційна хвороба, яка вперше виявлена в 2019 році в місті Ухань, Китай.
API (Application Programming Interface)	–	набір чітко визначених методів для взаємодії різних компонентів
СКБД	–	система керування базами даних
IT (Information Technology)	–	інформаційні технології
JDBC (Java Database Connectivity)	–	промисловий стандарт взаємодії Java-застосунків з різними СУБД
URL (Uniform Resource Locator)	–	стандартизована адреса певного ресурсу
MVC (Model-View-Controller)	–	шаблон проектування типу "Модель-Представлення-Контролер"
XML (Extensible Markup Language)	–	стандарт, що визначає правила мови опису інформації шляхом застосування тегів

ВСТУП

Формула-1 - чемпіонат світу по кільцевим автоперегонам на автомобілях з відкритими колесами. Слово "формула" означає список правил, яким повинна відповідати техніка для допуску до участі у змаганнях.

Розмови про заснування єдиного світового чемпіонату найшвидших машин ходили ще у 30-ті роки XX-століття. Завадою на шляху стала Друга світова війна, під час якої перегонам у світі не було місця. Чемпіонат засновано в 1950 році, а контроль над ним отримала сформована раніше організація FIA.

З тих часів техніка Формули-1 є провідною автомобільною розробкою, технології якої все частіше використовуються в дорожніх машинах. Саме після цього чемпіонату з'явились задньомоторна компоновка автомобіля, активна підвіска, безкаркасна конструкція, турбокомпресор, рекуперація енергії тощо. Технічні правила чемпіонату регулярно змінюються, що призводить до нових відкрить в сфері машинобудування.

Технічне забезпечення є дуже важливим фактором, але доповнює та повністю розкриває його пілот. Це фізично та морально підготована людина з видатними навичкам водіння складної та надшвидкої техніки, яка кожного разу ризикує своїм життям та має волю до перемоги. Пілоти Формули-1 завжди вважались найкращими гонщиками у світі, адже їм доводиться пілотувати та боротися на автодромах, де максимальна швидкість досягає 350 кілометрів за годину.

В кожній команді по 2 пілоти, метою яких є випередити свого напарника, а також усіх інших гонщиків. Боротьба пілотів є предметом обговорення спортивної спільноти, а найгостріші та найцікавіші з них - стають легендами. Чемпіони Формули-1 - світові зірки. Їх подвиги залишаються в історії, а їх імена - стають прозивним для людей, які полюбляють швидку їзду. Кожному відомі імена Міхаеля Шумахера, Льюїса Хемільтона, Айртона Сенни, бо вони - чемпіони.

В чемпіонаті Формули-1 приймають участь декілька команд(зазвичай 10), які використовують боліди власного дизайну та виконання. До їх складу входять

інженери високого рівня в сферах аеродинаміки, двигунобудування, термодинаміки та електроенергетики. Навіть якщо команда названа іменем якось гігантського машинобудівного концерну, це не означає що вона розміщена на їхніх заводах. Навпаки, це самостійна організаційна структура, яка своїми силами будує болід та займається його підтримкою та оновленням.

Чемпіонат світу у класі «Формула-1» відбувається щороку і складається з Гран-прі, або етапів, які проводяться на спеціально побудованих трасах, або підготовлених вулицях міста. Наприкінці сезону, за підсумками всіх гонок визначається переможець чемпіонату. У Формулі-1 змагаються як окремі пілоти, так і команди. Пілот-переможець отримує титул чемпіона світу, а команда-переможець отримує кубок конструкторів.[1]

Гран-прі - це не просто гонка у неділю. Під цим словом розуміють усі заходи, які відбуваються за декілька днів на конкретному автодромі. Зазвичай Гран-прі починається в четвер з різноманітних прес-конференцій пілотів та керівників команд, рекламних заходів та спілкуванням з вболівальниками. В п'ятницю проходять дві сесії тренувань, або як їх ще називають "вільні заїзди". В цей час команди мають змогу протестувати поведінку болідів, виконати фінальні налаштування та продумати стратегію. Субота починається з третього та останнього тренування, де команди здійснюють підготовку до кваліфікації. Кваліфікація призначена для формування стартових позицій для недільної гонки. Вона складається з 3 частин, де кожний пілот намагається показати якомога кращий результат. В неділю Гран-прі закінчується головною подією тижня - гонкою Формули-1.

Результати усіх сесій на трасі зберігаються самим чемпіонатом в базу статистики та публікується на сайті. Ці дані є дуже важливими для великої кількості людей: журналісти використовують результати для оцінки сил у чемпіонаті та підкресленні різного роду оновлень; букмекери використовують швидкість для підбору коефіцієнтів - кожна сесія динамічно впливає на їх збільшення або зменшення; інші команди оцінюють швидкість та потенціал своїх конкурентів.

На сьогоднішній день дуже просто отримати дані щодо попереднього Гран-прі, але так було не завжди. На початку чемпіонату, у 1950 році ще не існувало

електронних баз даних. Більш того, навіть час пілота на колі доводилось вимірювати ручними секундомірами, які завжди мають похибку в декілька десятих секунди. В ті часи приблизні числа зберігались у письмовому форматі та інколи публікувались у спеціалізованих виданнях та наукових працях на цю тематику. Таке становище до цих пір доставляє неприємностей авторам, які хочуть чітко та повно розповісти про ті часи, коли автоспорт тільки зароджувався.

Після того, як світ зіткнувся з пандемією COVID-19, стало зрозуміло, що не усі спортивні заходи можна буде проводити як раніше. Більшість видів спорту відмовились від глядачів на трибунах, або зменшили їх кількість в декілька разів. Такі рішення значно зменшили дохід переважної більшості видів спорту. Особливо негативно це відобразилось на малопопулярних змаганнях, для яких єдиним прибутком був продаж квитків. В цьому сенсі Формула-1 є винятком. Основну частину доходів чемпіонату складають спонсорські кошти та рекламні контракти, продаж прав на телевізійні трансляції, а також продаж різноманітної атрибутики. Усе це можна реалізовувати онлайн. Саме через це аудиторія Формули-1 стрімко зростає і все більше глядачів бажають ознайомитись з попередніми подіями чемпіонату. Найкращий спосіб отримати повну картину минулих років - через статистику.

Статистика прямо впливає на результати команди. В історії автоспорту є безліч прикладів, коли команда, проаналізувавши дані, робила вирішальні зміни в налаштуваннях або конструкції боліду. Для отримання дійсних покращень необхідно надати зручний та зрозумілий інструмент, який дасть змогу не тільки спеціалістам, але й пересічному глядачу провести аналіз. Складена база даних стане універсальним джерелом інформації. Вона не буде прив'язана до конкретного програмного продукту, а може використовуватись в різноманітних застосунках на будь-яких платформах та операційних системах.

Актуальність теми полягає у необхідності створення застосунку для відображення статистичних даних за всю історію Формули-1, який дасть змогу зручно їх групувати та аналізувати зацікавленим в цьому особам. Це дозволить авторам наукових та публіцистичних праць мати універсальне джерело інформації, а

також звичайним прихильникам чемпіонату розширити свої знання та прослідкувати тенденції розвитку техніки.

Об'єктом дослідження є інформаційні процеси системи групування та відображення статистичних даних.

Предметом є інформаційна система відображення статистики в контексті технології вилучення інформації з бази даних на персональному комп'ютері.

Мета дипломного проекту полягає у розробці застосунку для відображення детальних статистичних даних за всю історію чемпіонату.

Практичне значення дипломного проекту полягає у створенні програмного продукту, який безперебійно та за відсутності Інтернет-підключення отримати чітку та точну інформацію, а також компонування бази даних, яка матиме широкий спектр використання.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА ФОРМУЛИ-1 ТА АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ЗАСТОСУНКУ

Сучасні мови програмування здатні виконувати майже усі поставлені перед ними задачі. Для максимально ефективного проектування необхідно правильно обрати архітектуру майбутнього застосунку, інструменти написання коду та технології. Основний етап моделювання програмного продукту - створення алгоритму та визначення основного функціоналу.

1.1. Опис предметної області

Формула-1 - чемпіонат світу з кільцевих автоперегонів на болідах з відкритими колесами, заснований у 1950 році. За всю історію змагань можна згадати декілька разів, коли вони впливали на весь світ.

"Королева автоспорту" - це не тільки спорт. Це великий та серйозний бізнес, вартість якого за різними підрахунками становить більше 2 мільярдів доларів США. Чемпіонат став платформою як для незалежних конструкторів, так і для великих автомобільних концернів, які намагаються довести, що їхня техніка найшвидша.

Якби не Формула-1, світ би не знав такі назви як Alfa Romeo, Ferrari, McLaren, Ford тощо. Усі вони починали свій бізнес з перемог у світовій першості. Більшість з них й досі продовжують виступати та підтримують свій імідж.

Кожного року аудиторія чемпіонату зростає. Все більше нових уболівальників хочуть дізнатись про те, як він зароджувався та хто був найкращим у різні періоди історії. На мою думку, повна та детальна база даних дасть представлення про кожен з періодів, адже статистика висвітлює факти.

У Формулі-1 змагаються як пілоти, кожен сам за себе, так і команди. Тому майбутню базу даних можна уявно поділити на дві частини: дані про пілотів та дані про команди. В кожній команді за регламентом повинно бути по 2 пілоти.

Перегони відбуваються на чотирьох материках світу. В середньому проходять більше 20 гонок на десятках різних автодромів за сезон. Пілоти змагаються за бали, а конструктори отримують суму балів обох пілотів.

Формула-1 - це не лише гонка. Їй передують три сесії "вільних" заїздів, які по-суті можна назвати тренуванням. В п'ятницю та першу половину суботи пілоти та команди мають можливість протестувати оновлення на підібрати правильні налаштування боліду під кваліфікацію. Так як за час тренування пілоти не отримують жодних балів, а також той факт, що прямо не впливає на недільну гонку, то результати цих заїздів до бази даних можна не включати.

Після закінчення третього тренування та паузи в 2 години починається кваліфікація, яка визначає стартовий порядок пілотів в недільній гонці. Вона складається з 3 частин. З 20 пілотів, які приймають участь у першій частині, до останньої залишаються тільки 10, які і змагаються між собою за першу позицію. Кваліфікація є дуже важливим фактором перегонів та показником швидкості боліду, тому її результати необхідно включити в базу даних.

Закінчується етап чемпіонату світу гонкою. Це той самий момент, коли команди та пілоти показують світу свою підготовку та майстерність. По закінченню заїзду, перші 10 пілотів отримують бали, які зараховуються у чемпіонат пілотів, а конструктори отримують суму балів обох пілотів, які записуються у чемпіонат конструкторів.

Наприкінці сезону пілот, який заробив найбільшу кількість балів отримує кубок пілотів та звання чемпіона світу, а команда - кубок конструкторів.

1.2. Сучасні архітектури застосунків

Архітектури, або шаблони застосунків дають можливість глобально представити майбутній продукт, розділити його на модулі в межах заданого контексту. Така методика значно заощаджує час, адже вона являє собою основу для аналізу та дає змогу оцінити складену систему не створюючи її.

Одна з основних переваг архітектурного підходу - повторне використання. В цілому застосунки нерідко бувають схожі за своїми параметрами. Шаблон дозволяє повторно використовувати попередні напрацювання в декількох системах.

Початок архітектурі програмного забезпечення як концепції було покладено в науково-дослідній роботі Едсгера Даейкстри в 1968 році і Девіда Парнаса на початку 1970-х. Ці вчені наголосили, що структура системи ПО має важливе значення, і що побудова правильної структури - критично важливо. Популярність вивчення цієї області зросла з початку 1990-х років разом з науково-дослідними роботами з дослідження архітектурних стилів (шаблонів), мов опису архітектури, документування архітектури, і формальних методів.

У розвитку архітектури програмного забезпечення як дисципліни відіграють важливу роль науково-дослідні установи. Мері Шоу і Девід Герлан з університету Carnegie Mellon написали книгу під назвою «Архітектура програмного забезпечення: перспективи нової дисципліни в 1996 році», в якій висунули концепції архітектури програмного забезпечення, такі як компоненти, з'єднувачі (connectors), стилі і так далі. У каліфорнійському університеті інститут Ірвайна по дослідженню ПО в першу чергу досліджує архітектурні стилі, мови опису архітектури і динамічні архітектури.

Якщо зі старту процесу архітектурного проектування приділити занадто багато уваги опрацювання деталей майбутнього програми, але не досить скрупульозно пропрацювати логіку взаємодії його елементів, архітектор ризикує втратити образ архітектури і упустити контроль над керуванням програмним продуктом і його розвивається складністю, заплутавшись у незначущих деталях.

Архітектура повинна однозначно визначати структуру інформаційного продукту, що розробляється. Одна з основних асоціацій, що приходять на думку при

слові архітектура - структура. Паралелі, постійно проводяться між областю інформаційних технологій і будівництвом, мають місце на існування і для направлення архітектурного проектування. Якщо ви попросите колегу поверхнево описати архітектуру програмного забезпечення, з яким він працює, то, в 8 випадків з 10, він продемонструє схему / діаграму / модель, на якій будуть зображені структурні артефакти системи (об'єкти та зв'язку). Структурні характеристики архітектури проявляються багатьма способами в різних ситуаціях. Структурний елемент може бути цілою підсистемою, процесом, бібліотекою, базою даних, обчислювальним вузлом, системою в традиційному сенсі, готовим продуктом і так далі.[2]

Разом з визначенням структурних елементів кожна архітектура визначає взаємодії між ними. Саме характер і тип певних зв'язків забезпечує функціональність проектованої інформаційної системи.

Незважаючи на те, що архітектура визначає структуру і логіку взаємодії, вона не є домінуючим фактором. Область застосування, на яку поширюється вплив архітектури програмного забезпечення, обмежується значущими елементами, використання яких має тривалий і досить сильний вплив на автоматизує бізнес процеси.

Будучи дисципліною без чітких правил по шляху створення системи, проектування архітектури ПЗ все ще є сумішшю науки і мистецтва. Аспект «мистецтва» полягає в тому, що будь-яка комерційна система має на увазі наявність застосування або місії. Те, які ключові цілі має система, описується за допомогою сценаріїв як нефункціональні вимоги до системи, також відомі як атрибути якості, що визначають, як буде вести себе система. Атрибути якості системи включають в себе відмовостійкість, збереження зворотної сумісності, розширюваність, надійність, придатність до сервісного обслуговування (maintainability), доступність, безпеку, зручність використання, а також інші якості. З точки зору користувача програмної архітектури, програмна архітектура дає напрямок для руху і вирішення завдань, пов'язаних зі спеціальністю кожного такого користувача, наприклад, зацікавленої особи, розробника ПЗ, групи підтримки ПЗ, фахівця із супроводу ПЗ, фахівця з розгортання ПЗ, тестера, а також кінцевих користувачів. У цьому сенсі архітектура

програмного забезпечення насправді об'єднує різні точки зору на систему. Той факт, що ці кілька різних точок зору можуть бути об'єднані в архітектурі програмного забезпечення є аргументом на захист необхідності і доцільності створення архітектури ПЗ ще до етапу його розробки.[3]

Існує 10 найпопулярніших архітектурних шаблонів, які варто розглядати для застосунку відображення даних.

Багаторівневий шаблон використовується для структурування підзадач, які можливо розділити на групи підзадач, що знаходяться на певних рівнях абстракції. Кожен шар абстракції надає служби для більш верхнього шару. В сучасних інформаційних системах частіше за все використовуються 4 шари:

- Шар представлення. Саме тут знаходиться користувацький інтерфейс;
- Шар застосунку. Цей шар надає мультимедійні сервіси для усіх інших елементів застосунку;
- Шар бізнес-логіки. Повністю залежить від обраної предметної області, впроваджує як функціонал, так і обмеження;
- Шар доступу до даних. Виконує функції збереження даних, роботу з базами даних;

Кожен шар залежить тільки від шарів нижнього рівня та може самостійно існувати без шарів верхнього рівня. Такий шаблон використовується в застосунках загального призначення для персональних комп'ютерів.

Клієнт-серверний шаблон складається з двох частин: сервера та багатьох клієнтів. Основною функцією сервера є надання служб та сервісів підключеним до нього клієнтам. Клієнти запрошують послуги у сервера, а сервер їх надає. Найчастіше сервер знаходиться у постійному режимі "слухача", тим самим він постійно очікує підключення клієнтів. Клієнт та сервер повинні мати визначенні протоколи підключення та передачі даних. Даний шаблон широко використовується в онлайн-додатках, таких як електронна пошта, Інтернет-банкінг тощо.

Шаблон *"Master/Slave"* або *"ведучий-ведений"* також вимагає задіяння двох або більше учасників - ведучий та ведений. Ведучий компонент розподіляє задачі між веденими та прораховує кінцевий результат на основі даних, отриманих від ведених

компонент. В деяких системах статус компонент не визначено. В таких випадках система самостійно обирає ведучого з групи пристроїв, а всі інші автоматично стають веденими. Такий підхід вже багато років використовується в реляційних базах даних, а також в електронних пристроях.

Канали та фільтри. Такий шаблон стане корисним для систем, які працюють з потоками даних. В таких системах створюються окремі незалежні компоненти, які можна повторно використовувати для перетворення дискретних потоків даних від входу до виходу. Такі модулі називаються фільтрами. Ці фільтри зв'язані між собою комунікаційними каналами, які дають змогу передавати дані між компонентами. Дана архітектура використовується в системах, створених для односторонньої обробки даних, наприклад в компіляторах.

Шаблон посередника необхідний для структуризації розподілених систем з незв'язаними між собою компонентами. Такі компоненти можуть взаємодіяти між собою через віддалений виклик служби. Тобто модуль посередника відповідає за взаємодію усіх інших модулів. Робота системи влаштована наступним чином: сервер розміщує свої сервіси та служби у посередника, а клієнт, який бажає мати доступ до сервера, перенаправляється до посередника, а той в свою чергу перенаправляє до бажаної служби в своєму реєстрі.

Одноранговий шаблон визначає систему як сукупність рівноправних компонентів(піри), кожен з яких може виступати як сервером, так і клієнтом. Такий підхід широко використовується в проектуванні мереж. Для одноранговых систем характерна відсутність централізованого управління.

Подієво-орієнтована архітектура існує для створення, виявлення та реагування на події. В основному такий шаблон складається з 4 компонентів: джерело події, слухач події, канал та шина подій. Джерела розміщують події для певних каналів на певних шинах. Слухачі встановлюються на певні канали. Така архітектура значно ускладнює проектування застосунку, а також забирає більше часу для виявлення проблем або помилок та їх усунення.

Шаблон "Дошка" використовується для додатків, для яких відсутні чіткі детерміновані рішення. Він складається з 3 основних компонентів:

- "Дошка" - це структурована глобальна пам'ять, яка містить об'єкти із простору можливих рішень;
- Джерело знання - спеціальні модулі, які мають своє власне представлення щодо вирішення завдання;
- Компоненти управління - налаштування та виконання модулів.

Усі компоненти мають доступ до дошки. Компоненти можуть створювати нові об'єкти, які в подальшому буде додано туди. Така архітектура частіше за все застосовується до систем штучного інтелекту, таких як визначення транспортних засобів, розпізнавання голосу тощо.

Інтерпретатор використовується для розробки системи, яка перетворює додатки, написані на одній мові, на іншу мову програмування. Також він може перетворювати окремі ділянки скомпільованого коду. Використовується для запитів до баз даних.

Шаблон "Модель-представлення-контролер" або "MVC" є одним з найпопулярніших та найефективніших архітектурних представлень сьогодні. Він розділяє функціональність застосунку на компоненти трьох видів:

- Модель(Model) - містить дані застосунку, а також майже весь функціонал.
- Представлення(View) - містить ті дані, які будуть показані користувачу. Також взаємодія з користувачем відбувається саме через цей компонент.
- Контролер(Controller) - виступає в якості посередника між моделлю та представленням. Основна функція - відслідковування зміни стану та обробка даних від користувача.

Такий підхід введено задля розмежування представлення інформації від способів її представлення та прийняття від користувача. Також це сильно допомагає під час виправлення помилок, адже якщо виникла помилка під час роботи з користувацьким інтерфейсом, то одразу зрозуміло де її шукати.

Саме ця архітектура краще всього підходить для проектування додатків, де основний акцент робиться на взаємодію з користувачем, графічний інтерфейс та маніпуляції з даними.

1.3. Мова програмування Java. Бібліотека JavaFX

Java - об'єктно-орієнтована мова програмування, розроблена Sun Microsystems в 1996 році. З того часу вона стала однією з провідних мов програмування високого рівня в світі. Таке досягнення обумовлене великою кількістю важливих унікальних особливостей, які притаманні Java. Основним принципом цієї мови є підхід WORA - write once, run anywhere або "пиши один раз, запускай будь-де". Це дає можливість використовувати Java-додатки на будь-яких платформах, якщо вона підтримує середовище виконання Java.

Java є відносно простою для освоєння як для новачків, так і для досвідчених програмістів, адже її синтаксис дуже схожий на C/C++.

Дуже важливою перевагою Java над іншими є автоматична робота з пам'яттю. В ній реалізовано автоматичний механізм очищення сміття, який звільняє розробника від додаткових обов'язків.

Дана мова не просто виконує усі команди розробника, але й допомагає йому. В Java реалізовано механізми для пошуку та виявлення помилок та виключень, таких як вихід за межі масиву, невідповідність типів, які можуть негативно сказатись на роботу застосунку. Мова надає потужний апарат виключень, але і потребує від розробника їх оброблень.

Підтримка аплетів вважається однією з найважливіших функцій Java. Аплет - невеликий програмний продукт, який функціонує в межах комплексної системи та задля загальної безпеки відокремлений у власне середовище. Такі аплети можуть виконуватись як у Web-браузері, так і у звичайних додатках.

На самому початку Java проектувалась як мова для розподіленого проектування. Так як програмна система складається з окремих модулів, її можуть створювати декілька людей, що знаходяться в різних місцях. Java має вбудований механізм спільного використання даних та додатків декількома комп'ютерами, що підвищує швидкість проектування та ефективність праці.

Для максимально ефективного використання ресурсів процесора Java підтримує багатопотоковість. Потоки - це найменша одиниця обробки даних. Вони

використовують одну й ту саму область пам'яті, що дозволяє швидко переключатись між ними. Цей механізм є дуже важливим в роботі з графікою та анімаціями.

Мова Java - найпопулярніша програмування бізнес-додатків. Бізнес-додатки - це багатофункціональні програмні системи і комплекси, призначені для автоматизації ключових бізнес-функцій і процесів всередині компанії. Наявність таких додатків підвищує ефективність ключових бізнес-процесів підприємства, скоротити терміни пошуку і обробки інформації, економити матеріальні та людські ресурси, автоматизувати типові дії в рамках виконання процесів тощо. Завдяки великій спільноті розробників на Java, в мережі існують сотні бібліотек та фреймворків для будь-яких задач. На тему Java створено декілька тисяч посібників, курсів, уроків, порталів на більшості мовах світу.

Розробники на Java самостійно модифікували систему безпеки додатків. Так, є можливість додатково захистити класи за допомогою цифрового підпису. Доступ до таких класів надається тільки при повному підтвердженні довіри. Java - одна з найбільш надійних мов програмування світу. Програми на Java працюють при будь-яких умовах, а компілятор вказує на помили ще до початку роботи застосунку.

Завдяки динамічності та адаптованості Java продовжує працювати в постійно змінюваному середовищі. Попередньо написані додатки можна пристосувати до нового середовища шляхом додання нових бібліотек, об'єктів, методів. Додатки на Java можуть підключати об'єкти з мережі та відкривати до них доступ. Підтримується широка бібліотека для передачі даних по самих популярних протоколах, таких як FTP, HTTP, TCP/IP тощо.

Java має вбудовану підтримку найпопулярніших баз даних. Реалізувавши необхідний інтерфейс, додатки на Java можуть взаємодіяти з MySQL, Postgres, Oracle для зберігання, редагування та інших операцій над даними.

Мова програмування Java дає можливість проектувати та програмувати графічні інтерфейси взаємодії з користувачем як вбудовано, так і за допомогою сторонніх бібліотек та фреймворків, розроблених спільнотою. Вони розширюють можливості розробника, дозволяють максимально стилізувати кожен елемент інтерфейсу під потреби користувача. Однією з таких бібліотек є JavaFX.

JavaFX - потужний інструмент для проектування графічного інтерфейсу користувача. Він дозволяє максимально гнучким чином стилізувати файли компоновки, представлені у форматі XML, та надати їм зовнішнього вигляду за допомогою CSS, який широко використовується у веб-розробці та знайомий кожному розробнику. Ця бібліотека також відмінно оптимізована для роботи з 3D-графікою, аудіо, відео. Реалізована можливість створення анімацій, регламентувати їх вигляд, швидкість, напрямок тощо.

Якщо звичайний інтерфейс потрібно програмувати в коді та уявляти в голові, то JavaFX має зручний інструмент під назвою SceneBuilder, який дозволяє наглядно та графічно розміщувати елементи керування у вікні інтерфейсу. SceneBuilder - безкоштовний, має приємний та зрозумілий дизайн, повністю інтегрований у JavaFX та дозволяє редагувати попередньо створений інтерфейс у режимі реального часу.

В JavaFX вже вбудовані примітивні елементи керування, такі як текстове поле, кнопка, таблиця, різного роду гістограми та діаграми. Це дозволяє заощадити час розробника, який він витратив би на відтворення необхідних йому примітивів. Такі елементи можна об'єднувати у групи, розміщувати симетрично, створюючи композицію інтерактивних компонентів. Цей фреймворк підтримує архітектуру MVC, описану вище. Такий підхід дасть можливість розмежувати графічні елементи інтерфейсу, бізнес-логіку застосунку та контроль над взаємодією з користувачем.

1.4. Система керування базами даних MySQL

MySQL - одна з найпопулярніший систем керування базами даних(СКБД) сьогодні. Це обумовлено відносною простотою у використанні, широким функціоналом та сильною системою безпеки за замовчуванням.

Дана система представляє собою програмне забезпечення з відкритим вихідним кодом, що робить її безкоштовною та готовою до модифікацій. MySQL - це лише одна з багатьох СКБД для роботи з мовою структурованих запитів SQL(Structured Query Language). Головною перевагою MySQL над іншими є те, що вона без проблем працює з інтерфейсом API. За допомогою цього програмного забезпечення розробник

може легко отримати доступ із свого застосунку до СКБД на усіх популярних мовах програмування.

В Java існує MySQL Java Database Connector(JDBC) який і забезпечить підключення. Цей драйвер написано на Java і він не потребує встановлення додаткових сторонніх бібліотек. Саме підключення відбувається максимально просто: щоб отримати з'єднання до серверу з базою даних, необхідно мати URL-адресу серверу, ім'я користувача, пароль та саму базу даних. Java дозволяє отримувати велику кількість підключень до різноманітних баз даних одночасно.

В даний час існують версії СКБД для більшості поширених комп'ютерних платформ. Це говорить про те, що вам не нав'язують певну операційну систему. Ви самі можете вибрати, з чим працювати, наприклад з Linux або Windows, MacOS або Solaris, але навіть в разі заміни ОС ви не втратите свої дані і вам навіть не знадобляться додаткові інструменти для їх перенесення.

1.5. Створення бази даних

Необхідно приступити до побудови таблиць та складання зв'язків між ними. Назви таблиць та усі інші дані буде записано англійською мовою для уникнення конфліктів з різними системами кодувань.

Проаналізувавши предметну область, можемо виділити такі сутності:

- сезон;
- гонка;
- кваліфікація;
- автодром;
- пілоти;
- команди;
- чемпіонат пілотів;
- чемпіонат команд(конструкторів);

Ці сутності буде реалізовано у вигляді таблиць, дані в яких будуть змінюватись нечасто та незначно. Також буде введено декілька таблиць, дані в яких змінюватимуться досить часто, адже чемпіонати продовжуються. Кожна таблиця матиме унікальний ідентифікатор - первинний ключ, який однозначно визначатиме певну частину статистичної інформації.

Таблиця *"seasons"* містить інформацію не про окремі деталі, а про весь сезон в цілому. Вона матиме два поля: year(рік, коли сезон відбувся) та url(посилання у форматі URL на статтю про відповідний сезон).

Таблиця *"races"* постачає інформацію про конкретні перегони. Складається з наступних полів: raceId(ідентифікатор гонки), year(рік, коли гонка відбулась), round(порядковий номер гонки в сезоні), circuitId(ідентифікатор автодрому), name(назва гонки), date(дата, коли гонка відбулась), time(точний час початку гонки), url(посилання у форматі URL на статтю про відповідну гонку).

Таблиця *"circuits"* необхідна для отримання інформації про автодроми, на яких проводились етапи чемпіонату світу. Вона містить наступні поля: circuitId(ідентифікатор автодрому), circuitRef(текстовий ідентифікатор автодрому), name(назва автодрому), location(місто розташування), country(країна розташування), lat(широта розташування автодрому), lng(довгота розташування автодрому), alt(висота розташування автодрому), url(посилання у форматі URL на статтю про відповідний автодром).

Таблиця *"qualifying"* постачає інформацію про кваліфікаційні сесії до перегонів. Інформація по стовпцях: qualifyId(ідентифікатор кваліфікації), raceId(ідентифікатор гонки), driverId(ідентифікатор пілота), constructorId(ідентифікатор команди), number(персональний номер пілота), position(позиція за підсумками кваліфікації), q1(найкращий час пілота в першому сегменті), q2(найкращий час пілота в другому сегменті), q3(найкращий час пілота в третьому сегменті).

Таблиця *"drivers"* містить інформацію про пілотів: driverId(ідентифікатор пілота), driverRef(текстовий ідентифікатор пілота), number(персональний номер пілота), code(код пілота, який відображається на телеметрії), forename(прізвище

пілота), surname(ім'я пілота), dob(дата народження пілота), nationality(національність пілота), url(посилання у форматі URL на статтю про пілота).

Таблиця *"driverStandings"* зберігає кубки пілотів наступним чином: фіксуються бали та позиція пілота у заліку після кожного етапу. Кубком пілотів вважається розташування гонщиків у заліку на момент останньої гонки. Інформація по стовпцях: driverStandingsId(ідентифікатор заліку пілотів), raceId(ідентифікатор гонки), driverId(ідентифікатор пілота), points(кількість балів на момент даної гонки), position(позиція у заліку на момент даної гонки), positionText(позиція у заліку в текстовому форматі), wins(кількість перемог на момент даної гонки).

Таблиця *"constructors"* містить дані про команди чемпіонату. Вона складається з таких полів: constructorId(ідентифікатор команди), constructorRef(текстовий ідентифікатор команди), name(назва команди), nationality(національність команди), url(посилання у форматі URL на статтю про команду).

Таблиця *"constructorStandings"* зберігає інформацію про командні кубки таким самим чином, як і кубки пілотів: constructorStandingsId(ідентифікатор командного заліку), raceId(ідентифікатор гонки), constructorId(ідентифікатор команди), points(кількість балів на момент даної гонки), position(позиція у заліку на момент даної гонки), positionText(позиція у заліку в текстовому форматі), wins(кількість перемог на момент даної гонки).

Таблиця *"constructorResults"* необхідна для отримання результату конструктора після певної гонки. Вона складається з 5 полів: constructorResultsId(ідентифікатор результату), raceId(ідентифікатор гонки), constructorId(ідентифікатор команди), points(кількість балів за гонку), status(статус, може приймати значення null або "D" - дискваліфікація).

Таблиця *"status"* дасть змогу отримувати уточнюючі пояснення та причини логічних невідповідностей даних. Дана таблиця містить всього два поля: statusId(ідентифікатор статусу) та status(прични сходу або значного відставання від лідера).

Таблиця *"pitStops"* зберігає дані про зупинки на зміну шин, ремонт або інші технічні зупинки по ходу гонки. Її поля: raceId(ідентифікатор гонки),

driverId(ідентифікатор пілота), stop(номер зупинки), lap(коло, на якому зупинився пілот), time(час зупинки), duration(тривалість зупинки), miliseconds(час зупинки в мілісекундах).

Таблиця *"lapTimes"* постачає дані про найкращий час пілота в гонці. Вона містить наступні поля: raceId(ідентифікатор гонки), driverId(ідентифікатор пілота), lap(номер кола), position(позиція в гонці), time(час), miliseconds(час в мілісекундах).

Таблиця *"results"* - основна таблиця бази даних. Вона містить інформацію про кожну гонку, яка відбулась з 1950 року в рамках чемпіонату світу, а інші таблиці систематизують дані, об'єднуючи перегони в чемпіонати, пілотів у команди тощо. Інформація по стовпцям: resultId(ідентифікатор результату), raceId(ідентифікатор гонки), driverId(ідентифікатор пілота), constructorId(ідентифікатор команди), number(номер пілота), grid(позиція на старті), position(позиція на фініші), positionText(позиція в текстовому вигляді), positionOrder, points(кількість балів), laps(кількість пройдених кіл), time(час або відрив від лідера), miliseconds(час в мілісекундах), fastestLap(номер кола з найкращим часом), rank(найкращий час в порівнянні з іншими пілотами), fastestLapTime(найкращий час), fastestLapSpeed(середня швидкість протягом найкращого кола), statusId(ідентифікатор статусу).

Після призначення відповідним полям властивості первинного та вторинного ключів, вибору типів даних та визначення зв'язків, отримуємо бажану базу даних, вигляд якої зображено на рисунку 2.1:

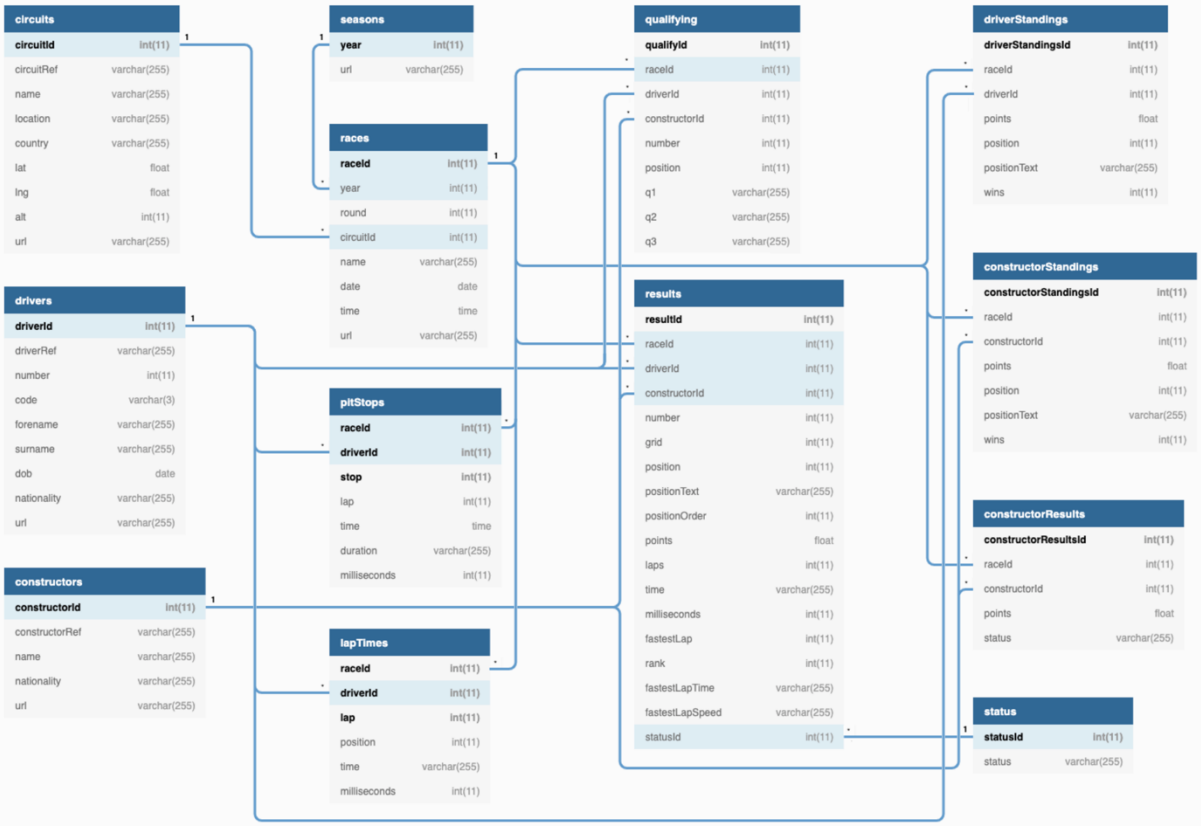


Рис. 2.1. ER-діаграма бази даних

Розроблені таблиці названо згідно призначення. Всі назви таблиць є повними, однозначними та повністю відповідають загальноприйнятим стандартам. В якості строкових типів даних використовується тип "varchar", так як він краще працює з пам’яттю та може зберігати як літерні і числові, так і літеро-числові рядки. Цілочисловий тип "int" є оптимальним для чисел, так як статистичні значення, що зберігаються в таблиці, не перевищують його максимально допустиме встановлене значення.

Висновки до розділу 1

Вибрана предметна область є дуже глибокою, а отже має багато статистичних параметрів, які потрібно висвітлити в таблицях бази даних. Більшість параметрів є числовими, але найважливіші продубльовано в строковому форматі.

Вибір архітектури майбутнього програмного забезпечення є дуже важливим першим кроком. Адже саме вона визначає функціональність застосунку, швидкість його роботи, захищеність та складність написання. Для мого дипломного проекту найкраще підходить архітектура "MVC" тому, що через модульність структури вона полегшує розробку та тестування окремих компонентів системи, дозволяє розширити або повністю змінити функціональні особливості без переписування усього коду.

Наступним кроком стає вибір мови програмування, яка підтримує обрану раніше архітектуру. Найкращим вибором стане та мова, яка максимально використовує можливості шаблону та дозволить динамічно модифікувати усі компоненти. Мова високого рівня Java не тільки підходить по цим параметрам, але ще й по своїм внутрішнім якостям. Додатки на Java можна запускати на різних системах, вони підтримують багатопотоковість та роботу з базами даних. Наявність бібліотеки JavaFX дозволить спроектувати зручний графічний інтерфейс для ефективної взаємодії застосунку з користувачем.

Вибір СКБД визначає яким чином розробник буде керувати даними. На мою думку, MySQL є найкращим варіантом. Вона проста у використанні, безкоштовна, з високим рівнем безпеки та підтримкою масштабованості. Через драйвер MySQL JDBC створена база даних легко підключиться до основних програмних модулів. База даних складена максимально точно, з урахуванням усіх норм та правил. В таблицях представлена відкрита статистика чемпіонатів, починаючи з 1950 року.

РОЗДІЛ 2
РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ВІДОБРАЖЕННЯ СТАТИСТИЧНИХ ДАНИХ

2.1. Структура застосунку

Дана ІС складається з алгоритму, який відповідає за вилучення та видачу даних, графічного інтерфейсу користувача та бази даних, розміщеної на локальному сервері. Для налаштування окремих елементів інтерфейсу будуть використовуватись каскадні таблиці стилів CSS, які зазвичай використовуються у веб-розробці разом з HTML.

Сам алгоритм буде розбито на декілька класів, кожен з яких матиме власне призначення. Така концепція буде повністю відповідати шаблону проектування MVC, обраному на стадії аналізу архітектур програмного забезпечення.

Головний клас Main буде відповідати за запуск застосунку, створення головного вікна інтерфейсу та завантаження в нього елементів керування з файлу розмітки формату XML.

Ініціалізація застосунку відбуватиметься автоматично. Фреймворк JavaFX запустить метод init(), який і відповідає за ініціалізацію, одразу після наслідування головним класом класу Application. Після цього запускається метод start() - головна точка входу у всіх додатках на JavaFX.

Метод main(), який зазвичай виконує роль стартової точки кожної програми на Java, в даному випадку ігнорується через наявність методу start().

Його наявність рекомендована тому, що деякі інтегровані середовища розробки працюють некоректно з бібліотекою JavaFX та потребують наявність цього методу. Якщо start() залишився нерозпізнаним, його функції виконує main(), передаючи необхідні для роботи застосунку параметри класу Application.

Таким чином застосунок починає свою роботу. Середовище виконання фреймворку входить в режим очікування команди завершення роботи програми. Після того, як такий сигнал надійшов(закриттям усіх вікон інтерфейсу або іншим способом), це середовище викликає метод stop() і застосунок повністю зупиняє свою роботу.

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("fxml/sample.fxml")))
;
        primaryStage.setTitle("F1 Statistics");
        primaryStage.setScene(new Scene(root, 1280, 720));
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2.2. Розробка основної частини застосунку

2.2.1. Розгортання бази даних

Для розгортання необхідно створити локальний сервер на комп'ютері та налаштувати стабільне підключення до нього, а також розмістити на ньому базу даних. Створення локального сервера відбуватиметься за допомогою програмного забезпечення MAMP.

MAMP за лічені секунди встановлює локальне серверне середовище на Вашому комп'ютері з системою ОС Mac або Windows. MAMP - безкоштовний і простий в установці застосунок. MAMP не ставить під загрозу існуючу установку Apache в системі. Ви можете встановити Apache, Nginx, PHP і MySQL без запуску скрипта і зміни конфігураційних файлів. Якщо MAMP більше не потрібен, просто видаліть папку MAMP і все повернеться в початкове положення (тобто MAMP нічого не змінить в системі).[4]

Компанія MAMP GmbH - розробник застосунку, пропонує дві версії програмного забезпечення - платну та безкоштовну. Платна версія MAMP Pro має усі функції звичайної версії, а також на застосунок розширений інтерфейс, службу підтримки, інтегрований текстовий редактор тощо. Для виконання дипломного проекту необхідно та достатньо використовувати безкоштовну версію застосунку.

Після завантаження та встановлення програмного забезпечення, наступним кроком стане вказання параметрів локального сервера. Ім'я хосту та ім'я користувача залишимо за замовчуванням - localhost та root відповідно. Порт підключення за замовчуванням встановлено 8889. Це стандартне значення, яке використовує MAMP. Воно обумовлене тим, що на персональному комп'ютері можуть бути запущені інші сервери з таким самим портом, і, для уникнення порушення зв'язку та помилок у роботі, усі сервери подібного роду повинні задіяти різні порти. В нашому випадку ніяких інших локальних серверів немає, тому в налаштуваннях повернемо загальноприйняте значення номера порту для MySQL - 3306. На останньому етапі обов'язковим моментом є вказання паролю до свого облікового запису.

Графічний інтерфейс застосунку MAMP є дуже простим та інтуїтивно зрозумілим. Натиснувши на кнопку "Start" застосунок запустить декілька локальних серверів, одним з яких і буде MySQL. Про успішне налаштування свідчать індикатори, розміщені вище.

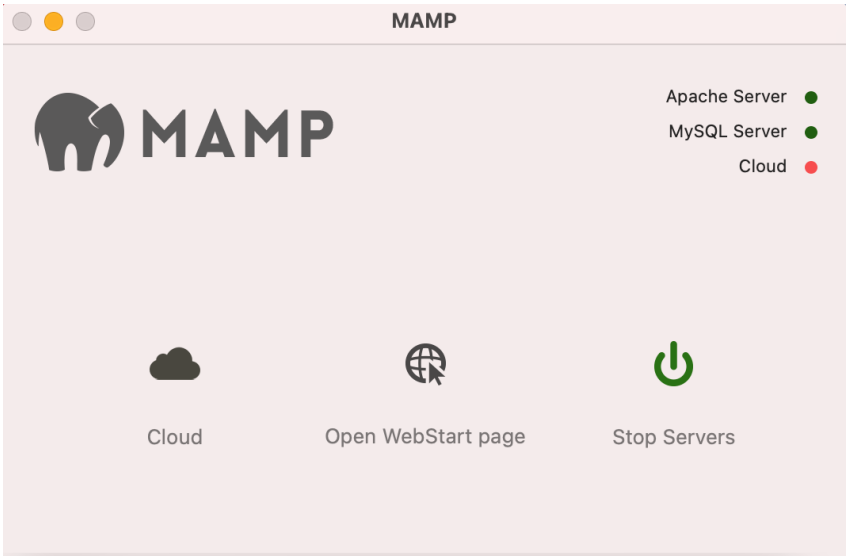


Рис. 2.1. Головне вікно застосунку MAMP

Після запуску локального серверу, у веб-переглядачі за замовчуванням відкриється сторінка, що засвідчує успішний початок роботи. Вона також містить подальші інструкції для початківців, стрічку новин та декілька корисних посилань. Сама сторінка розміщена на локальному сервері("localhost/MAMP"), що також є сигналом нормальної роботи застосунку.

Для розміщення бази даних на сервері існує велика кількість способів та інструментів. Такі завдання виконує будь-яка сучасна СКБД. Аби уникнути встановлення стороннього програмного забезпечення та заощадити дисковий простір на робочій машині, скористуємось все тією ж самою сторінкою, яку відкрив MAMP після запуску серверів. На ній, серед багатьох інших, розміщена секція "MySQL" з усіма необхідними даними нашого серверу.

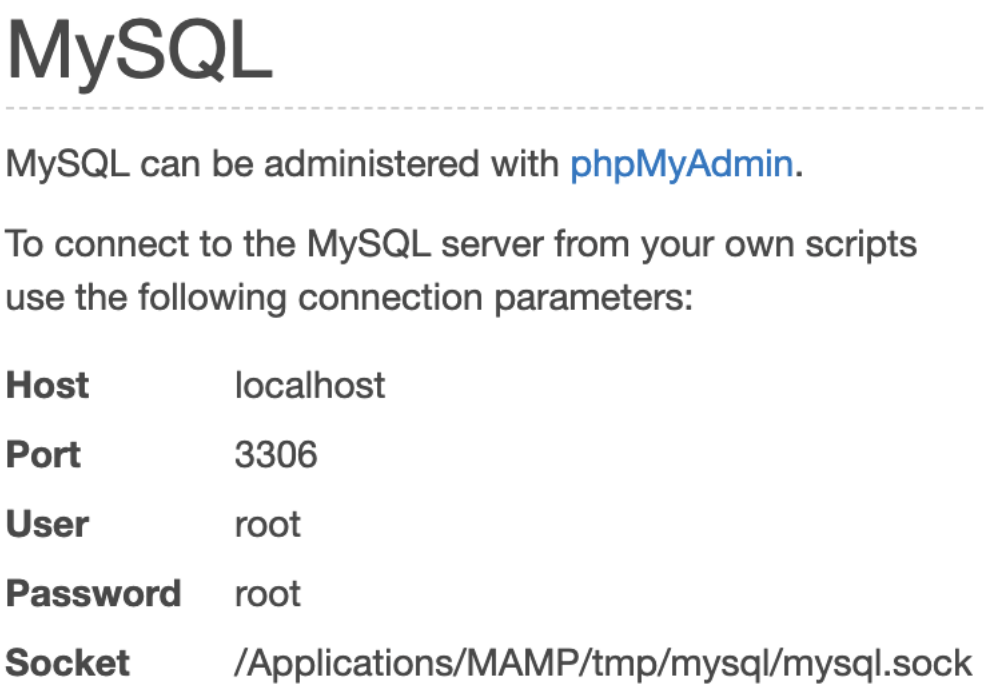


Рис. 2.2. Секція "MySQL" веб-сторінки

Запис "MySQL can be administered with phpMyAdmin" з англійської мови перекладається як "MySQL можна керувати за допомогою phpMyAdmin"

phpMyAdmin - це безкоштовний програмний засіб, написаний на PHP, призначений для управління адмініструванням MySQL через Інтернет. phpMyAdmin підтримує широкий спектр операцій на MySQL та MariaDB. Часто використовувані операції (управління базами даних, таблицями, стовпцями, відносинами, індексами, користувачами, дозволами тощо) можна виконувати через користувальницький інтерфейс, тоді як у вас все ще є можливість безпосереднього виконання будь-якого оператора SQL.

phpMyAdmin постачається з широким колом документації, і користувачі можуть оновити спеціалізовані вікі-сторінки, щоб поділитися ідеями та інструкціями щодо різних операцій. Команда phpMyAdmin спробує допомогти, якщо користувач зіткнетеся з якоюсь проблемою.

phpMyAdmin також дуже глибоко задокументований у книзі, написаній одним із розробників - «Освоєння phpMyAdmin для ефективного управління MySQL», яка доступна англійською та іспанською мовами.

Щоб полегшити використання широкому колу людей, phpMyAdmin перекладається на 72 мови та підтримує мови LTR та RTL.

phpMyAdmin - це зрілий проект зі стабільною та гнучкою базою коду; Даний продукт вже більше 15 років на ринку. Для нього можна знайти велику кількість навчальної літератури, відео- та аудіо-уроки, професійні розширення, а також обговорення в мережі та теми на форумах.

Проект phpMyAdmin є членом організації Software Freedom Conservancy. SFC - це неприбуткова організація, яка допомагає просувати, вдосконалювати, розробляти та захищати проекти Free, Libre та Open Source Software (FLOSS).[5]

phpMyAdmin надає не тільки графічний інтерфейс, але й підтримку більшості особливостей MySQL, імпорт та експорт баз даних, керування декількома серверами одночасно, створення графіків та діаграм для баз даних тощо.

Постійна підтримка та модернізація системи безпеки розробниками phpMyAdmin не дає зловмисникам отримати доступ до приватних даних шляхом завантаження шкідливого програмного забезпечення.

Натиснувши на посилання, зображене на рис. 2.2, ми потрапляємо на головну сторінку СКБД phpMyAdmin(рис. 2.3).

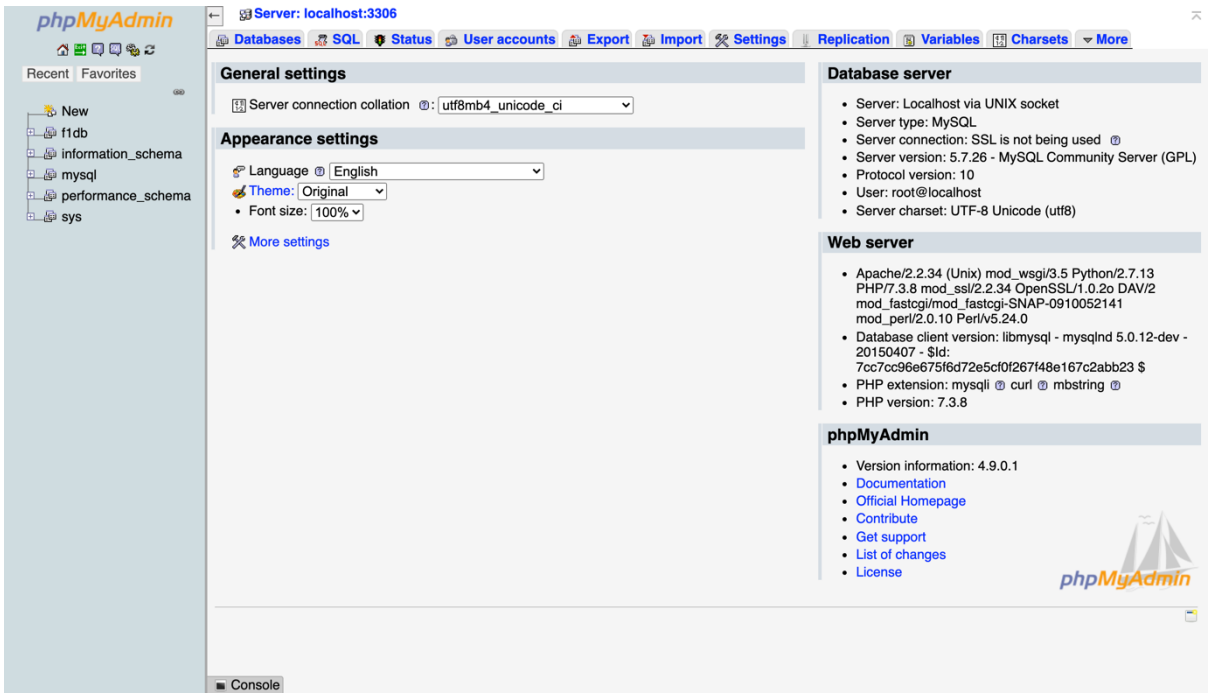


Рис. 2.3. Головна сторінка СКБД phpMyAdmin

Як було зазначено раніше, головною метою використання даного веб-застосунку є розгортання бази даних на локальному сервері, тому весь інший функціонал на цьому етапі нам не знадобиться.

Для завантаження бази даних з файлу на сервер необхідно створити порожню базу даних та наповнити її таблицями. Перейдемо на вкладку "SQL", де можна задавати та запускати SQL-запити до серверу.

Далі переходимо на вкладку "Import", яка призначена для завантаження бази даних з файлу формату, який зберігає код, написаний на мові SQL. Обравши необхідну базу даних зі списку на сервері, натискаємо кнопку "Go".

Таким чином, після усіх виконаних операцій, ми отримали базу даних, розміщену на локальному сервері. Для вилучення даних без наявності відповідного застосунку можна використовувати phpMyAdmin.

Локальний сервер також можна використовувати в подальшому для розміщення на ньому інших джерел даних, які вимагають для своєї роботи використання клієнт-серверної моделі.

2.2.2. Підключення до бази даних

Першочергово необхідно встановити підключення з сервером. В Java існує Java Database Connectivity(JDBC) API, яке входить в стандартну бібліотеку мови програмування. Воно дозволяє підключитись до будь-якої сучасної бази даних: Postgres, Oracle, SQL Server тощо. API дозволяє не тільки встановлювати з'єднання, але ще й виконувати запити та оброблювати результати, отримані з бази даних.

З боку MySQL необхідно встановити драйвер MySQL Connector, який і реалізує інтерфейс JDBC. Його можна знайти на офіційному сайті системи керування базами даних MySQL в розділі для розробників. Цей драйвер являє собою бібліотеку, написану на чистій мові Java, що є величезною перевагою над іншими інструментами, адже для його роботи не потрібно встановлювати додаткових підключень та сервісів. Завдяки середовищу програмування IntelliJ IDEA, MySQL Connector додається до проекту як зовнішня бібліотека, а шляхи до необхідних класів вказуються автоматично.

Механізм підключення стандартний. В Java входить бібліотека java.sql, яка містить клас "Connection", представлений у вигляді інтерфейсу.
Цей інтерфейс відкриває підключення до певної бази даних. Усі виконані запити та повернуті результати знаходяться в контексті даного підключення. База даних, підключена таким способом, може надавати дані щодо своїх таблиць, синтаксису, процедур, властивостей цього підключення тощо.

З боку драйвера MySQL Connector існує клас DriverManager, який має усі необхідні методи для створення підключення. Основний метод "getConnection()" дозволяє підключитись до сервера. Він потребує для цього адресу сервера у форматі URL, ім'я користувача та його пароль. В результаті, метод повертає об'єкт класу "Connection", який і записується у змінну для подальшої роботи з підключенням.

Одразу після налагодження підключення важливо створити об'єкт класу Statement, який дозволить в подальшому виконувати SQL-запити через Java-код. Для цього оголошуємо змінну даного класу та зберігаємо в ній значення, яке повертає метод "createStatement()" класу Connection.

```
Connection connectionTmp;
Statement statementTmp;
connectionTmp = DriverManager.getConnection("jdbc:mysql://localhost:3306/f1db?serverTimezone=CET", "root", "root");
statementTmp = connectionTmp.createStatement();
```

Таке підключення знаходитиметься як в Model, так і в Controller головної сторінки застосунку. Це зумовлено специфікою структури застосунку. Розділення підключень дасть можливість паралельно працювати декільком модулям, а також уникнути помилок та конфліктів під час виконання запитів та отримання доступу до даних.

2.2.3. Розробка інтерфейсу користувача

Графічний інтерфейс користувача визначає як відбуватиметься взаємодія людини з додатком. Він визначає зручність використання застосунку, його функціональність та ефективність.

Для розробки даного застосунку будуть використовуватись технології проектування інтерфейсів. Головними складовими є FXML та CSS.

FXML - мова розмітки, створена на основі XML корпорацією Oracle ексклюзивно для JavaFX. Він дає можливість скомпонувати інтерфейс у декларативній формі, як це відбувається в HTML. Для розуміння цієї мови необхідно розглянути на чому вона базується.

XML - розширювана мова розмітки. Він використовується в основному для збереження та передачі даних. Цей формат є загальноприйнятим та рекомендований організацією W3C. Його часто можна зустріти для передачі даних по API, а, в деяких випадках він взагалі єдиний.

Як і більшість мов розмітки, структура коду поділяється на теги. Вони бувають відкриваючі та закриваючі. В кожному коді є елемент з якого починається документ. Теги підтримують атрибути, які дають можливість коригувати стандартні властивості елементів розмітки.

XML не є заміною HTML. Вони призначені для вирішення різних завдань: XML виконує завдання збереження та транспортування даних, фокусування на тому, що такі самі дані, HTML же виконує задачу відображення даних, фокусування на тому, як ці дані переглядають. Таким чином, HTML вимагає відображення інформації, а XML - транспортування інформації.[6]

Усе вищеперераховане справедливо і для FXML. Однією з основних особливостей формату є те, що він дозволяє відділити графічний інтерфейс від головного коду. Це допомагає очистити обидві частини застосунку.

Формат FXML дозволяє не тільки вказати які елементи керування будуть знаходитись у певному вікні, але й задати їм атрибути, прив'язати логіку, стилізувати їх. Для стилізації можна використовувати як вбудовані інструменти мови, так і звичний для всіх спосіб - каскадні таблиці стилів CSS.

CSS - формальна мова опису зовнішнього вигляду документу, написаному мовою розмітки(зазвичай HTML). Вона використовується для модифікації зовнішнього вигляду сторінки, а саме таких складових як колір, шрифт, розміщення блоків тощо.

Не дивлячись на те, що JavaFX має власний набір властивостей, вони дуже схожі на аналогічні в CSS. В обох випадках використовується концепція селектора.

Селектор визначає до якого елемента розмітки застосувати стильові властивості. До елементів можна звертатись по класу, тобто групі однотипних елементів(кнопки, поля тощо). Можна створювати власні класи користувача, об'єднуючи різнотипні елементи. В JavaFX існують також ідентифікатори елементів.

Визначившись з дизайном та створивши вікно, ми отримуємо статичний застосунок. Наступним кроком стане додавання функцій.

Кожне вікно має власний клас-контролер. Це окремий модуль, який відповідає за динамічні властивості інтерфейсу. Саме тут знаходяться основні методи та функції керування. До більшості елементів розмітки можна прив'язати слухачі, які після спрацювання активують вказаний метод. В такому класі обов'язково визначені поля розмітки, що знаходяться в файлі FXML.

Проектувати комплексні інтерфейси без наглядного представлення складно. Програмне забезпечення SceneBuilder надає зручний набір інструментів, а також графічне представлення майбутнього інтерфейсу.

Застосунок складається з 2 вікон: головного та табличного. Головне вікно застосунку виглядає так:

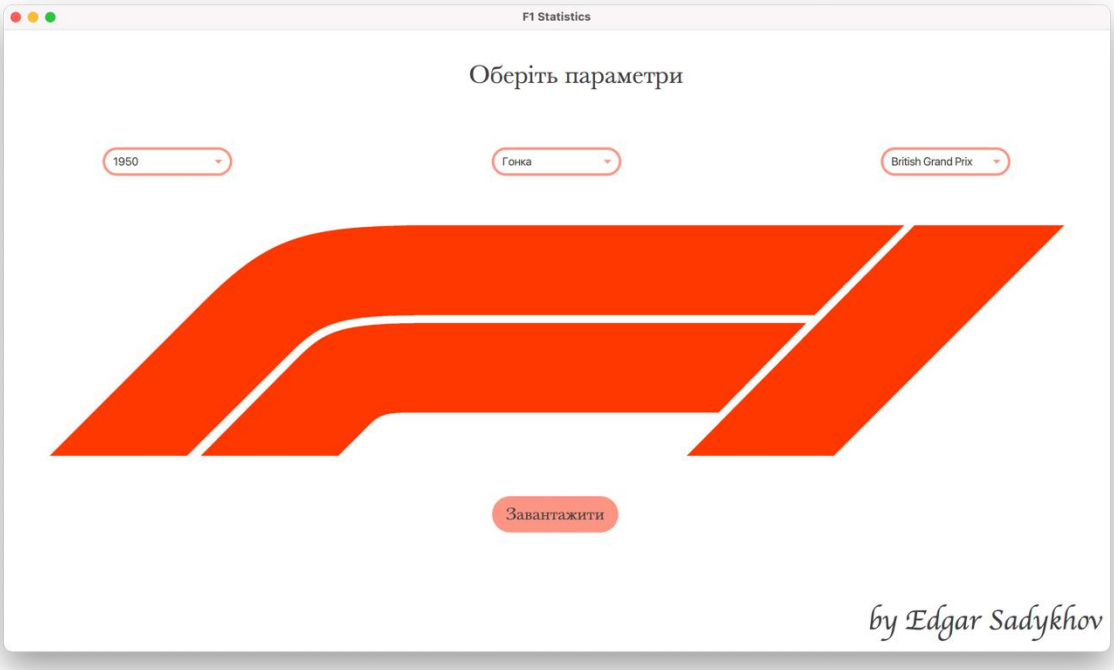


Рис. 2.4. Головне вікно застосунку

Назва вікна "F1 Statistics" задається в класі Main на етапі створення основної сцени. Саме вікно на протязі усього часу роботи застосунку не змінюватиметься. Основний принцип багатовіконності реалізовано через зміну сцен шляхом натискання на відповідні кнопки.

На задній план даного вікна встановлено зображення. Вузол ImageView бібліотеки JavaFX дає можливість працювати з усіма найрозповсюдженішими форматами картинок. Одним зі способів додання фото до сцени є через код застосунку. Спершу необхідно використати клас Image мови Java для завантаження зображень з жорсткого диску або мережі Інтернет.

Для додання зображень через FXML необхідно створити відповідний тег ImageView, в який буде вкладено непарний тег Image. Аби швидко отримувати доступ до бажаних фото був створений пакет media, де рекомендовано зберігати усі медіа-дані.

Після додавання зображення потрібно задати його розміри. Адже не кожне фото буде збігатись з розміром вікна. В такому випадку існує два варіанти: обрати зображення з роздільною здатністю нижчою за розмір вікна, або навпаки. Якщо

обрати перший варіант, то при збільшенні розмірів зображення, щоб відповідати розмірам сцени, воно почне втрачати якість та стане нечітким. В іншому випадку, при зменшенні завеликого зображення воно не втрачає пікселі, а отже виглядає так само.

В даному випадку, при розмірах вікна в 1280 на 720 пікселів, розмір фото складає 3840 на 2160 пікселів. Зображення такого розміру буде достатньо для відображення застосунку у режимі "На весь екран" для більшості моніторів. Важливим також є те, що при необхідності додати функції до головного вікна та збільшити його розмір, зображення можна буде підігнати до нового вікна без втрати якості.

```
<ImageView fitHeight="720.0" fitWidth="1280.0" layoutX="8.0" layoutY="17.0" pickOnBounds="true"
        AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
      <Image url="@../media/2180263.png" />
   </ImageView>
```

В правому нижньому куті знаходиться підпис автора. Текст наноситься на сцену за допомогою класу Label. Це компонент інтерфейсу, в який закладаються текстові дані(є підтримка символів та зображень). Label також підтримує додання стилів як через програмний код, так і через FXML. Використовуючи вкладені теги можна змінювати шрифт, кегль, колір заднього плану, колір тексту тощо. Label може тільки показувати текст або зображення, але не може отримувати фокус.

```
<Label layoutX="998.0" layoutY="631.0" text="by Edgar Sadykhov" AnchorPane.bottomAnchor="10.0" AnchorPane.rightAnchor="10.0">
    <font>
      <Font name="Apple Chancery" size="33.0" />
    </font>
  </Label>
```

Одразу під надписом "Оберіть параметри", який також реалізовано шляхом додання Label, знаходяться декілька випадаючих списків. Об'єкти класу ChoiceBox призначені для представлення користувачу вибору з відносно невеликої вибірки попередньо визначених пунктів. Після відкриття такого списку, на вибір користувачу буде запропоновано декілька варіантів, з яких можна обрати лише один.

JavaFX ChoiceBox є повним аналогом класу ComboBox, представленому у Java. Єдиною відмінністю є помітка, яка знаходиться навпроти того пункту списку, який обрав користувач. Обраний варіант буде відображатись після закриття списку.[7]

Такого роду списки можна створювати як порожніми, так і заповненими. Щоб додати дані до об'єкту ChoiceBox, потрібно обрати правильне представлення та структуру даних. Випадаючий список цього класу приймає дані, додані в об'єкт класу ObservableList.

ChoiceBox має реалізовану вбудовану підтримку динамічного додавання та видалення даних. Це означає, що маючи заповнений випадаючий список, можна його очищати та знову заповнювати в ході виконання застосунку. За необхідності, ChoiceBox дозволяє отримувати значення елементів, які він вміщає. При чому, можна одержати як значення тільки обраного елемента, так і увесь список загалом.

Найважливішою особливістю випадаючого списку ChoiceBox є можливість встановлення слухача. Така функція дає змогу отримувати інформацію про те, коли та який саме пункт обрав користувач. Як тільки такий момент виявлено, слухач виконує операції, визначені в коді програми.

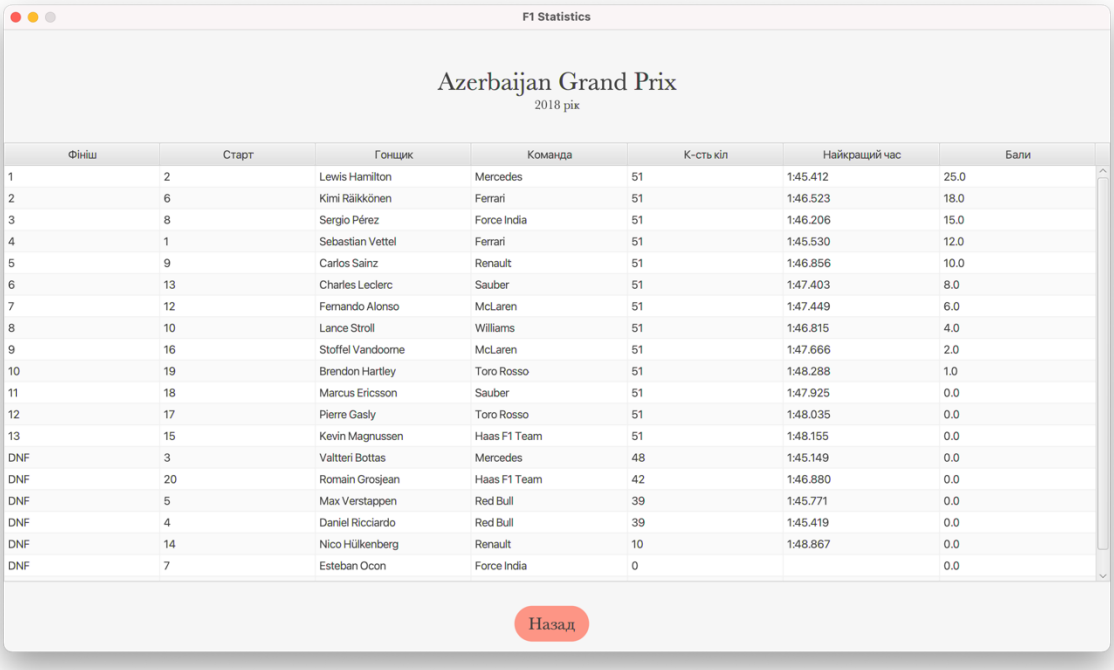
В центральній частину нижнього краю вікна знаходиться порожній елемент класу Label. Він слугує підказкою для користувача. Якщо користувач обрав некоректні параметри, слухач реагує на це записом у Label тексту про помилку та прохання до зміни цих параметрів. Як тільки дані обрано правильно, слухач знову реагує на це видаленням тексту цієї мітки.

Останнім елементом головного вікна є кнопка. Об'єкт класу Button бібліотеки JavaFX слугує для виконання певної дії після її натискання. Кнопка може містити в собі як текст, так і графічне зображення. Після того, як отримано сигнал про натискання кнопки, обробник подій виконує визначену операцію.

Як видно на рис. 2.4, зовнішній вигляд кнопки Button відрізняється від стандартного. Змінювати зовнішній вигляд елементів контролю можна за допомогою CSS. Звертаючись до одного або групи елементів цього класу, можливо змінювати колір, форму, розміри кнопки тощо. В даному випадку, об'єкт класу Button не тільки відповідатиме за відправлення даних з випадаючих списків до іншого класу, але ще й змінюватиме сцену у вікні. Після натискання кнопки, слухач отримає сигнал та активує метод ініціалізації нової сцени.

```
<Button fx:id="forwardBtn" layoutX="565.0" layoutY="540.0"
mnemonicParsing="false" onAction="#switchForward" text="Завантажити">
    <font>
      <Font name="Baskerville" size="20.0" /></font></Button>
```

Сцена відображення обраних даних виглядає так:



Фініш	Старт	Гонщик	Команда	К-сть кіл	Найкращий час	Бали
1	2	Lewis Hamilton	Mercedes	51	1:45.412	25.0
2	6	Kimi Räikkönen	Ferrari	51	1:46.523	18.0
3	8	Sergio Pérez	Force India	51	1:46.206	15.0
4	1	Sebastian Vettel	Ferrari	51	1:45.530	12.0
5	9	Carlos Sainz	Renault	51	1:46.856	10.0
6	13	Charles Leclerc	Sauber	51	1:47.403	8.0
7	12	Fernando Alonso	McLaren	51	1:47.449	6.0
8	10	Lance Stroll	Williams	51	1:46.815	4.0
9	16	Stoffel Vandoorne	McLaren	51	1:47.666	2.0
10	19	Brendon Hartley	Toro Rosso	51	1:48.288	1.0
11	18	Marcus Ericsson	Sauber	51	1:47.925	0.0
12	17	Pierre Gasly	Toro Rosso	51	1:48.035	0.0
13	15	Kevin Magnussen	Haas F1 Team	51	1:48.155	0.0
DNF	3	Valtteri Bottas	Mercedes	48	1:45.149	0.0
DNF	20	Romain Grosjean	Haas F1 Team	42	1:46.880	0.0
DNF	5	Max Verstappen	Red Bull	39	1:45.771	0.0
DNF	4	Daniel Ricciardo	Red Bull	39	1:45.419	0.0
DNF	14	Nico Hülkenberg	Renault	10	1:48.867	0.0
DNF	7	Esteban Ocon	Force India	0		0.0

Рис. 2.5. Вікно з даними

Зверху на сцені розміщені дві мітки, що показують які параметри було обрано на попередньому етапі. Верхня мітка містить дані про тип обраної інформації або події, а нижня - коли це відбулось. При деяких вихідних параметрах верхня мітка виконує функції нижньої, а та, в свою чергу, відсутня.

Під ними знаходиться спочатку порожня таблиця, без стовпців та комірок.

Об'єкти класу TableView являють собою табличні представлення в JavaFX-документах. Кожна таблиця складається зі стовбців, кожен з яких має чітко визначений тип даних, які він може вміщувати.

Представлення TableView може мати будь-який набір стовпців. Більш того, маючи на сцені лише один об'єкт TableView, можна реалізовувати декілька таблиць, змінюючи стовпці та дані в них.

У файлі розмітки FXML таблицю розміщено на всю ширину вікна, а через програмний код визначено те, що стовпці займають всю ширину таблиці та стають елементами однакового розміру.

В нижній частині сторінки розміщена кнопка "Назад" того самого класу Button JavaFX. Використовуючи CSS, змінено її зовнішній вигляд. До переліку змін входять колір межі кнопки, колір заднього фону, а також поведінка під час наведення на неї курсором. В такій ситуації колір заднього фону змінюється на білий, тим самим показуючи користувачу, що на неї наведено фокус.

Головною функцією цієї кнопки є зміна сцени. Після натискання, спрацьовує обробник подій, викликаючи метод повернення на головне вікно. Метод switchBackwards() розміщений у класі-контролері, який прив'язаний до даного FXML-файлу.

```
@FXML
    private void switchBackwards() throws Exception {
        Parent root =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource(View.FXML_SAMPL
E)));
        Stage stage = (Stage) backwardsBtn.getScene().getWindow();
```

```
        stage.setScene(new Scene(root, 1280, 720));
        stage.show();
    }
```

Перехід від головного вікна до табличного відбувається, коли користувач обрав бажані параметри запиту та натиснув на кнопку "Завантажити". Дані завантажуються у таблицю. Якщо користувач закінчив роботу з цим набором інформації, він натискає кнопку "Назад" та знову має можливість обрати параметри. Після кожного переходу дані в таблиці видаляються та замінюються на інші.

2.2.4. Розробка модуля вилучення даних

Розробка бізнес-логіки цього застосунку нерозривно пов'язана з процесом проектування користувацького інтерфейсу. Це обумовлене тим, що деякі елементи графічного інтерфейсу потребують представлення даних у певних форматах та структурах даних. Тому, незважаючи на те, що деякі алгоритми не є оптимальними загалом, вони необхідні для правильної роботи інтерфейсу.

Отримані дані будуть відображатись у вигляді таблиці. Вилучення з бази даних буде відбуватись у декілька етапів, тобто по одній колонці за раз. Для такого роду операцій скористаємось інтерфейсом ResultSet. Він дозволяє зберегти дані, які згенеровано виконанням SQL-запиту. ResultSet має вбудований курсор, який дає змогу переміщуватись між вилученими рядками. Цей курсор починається з першого рядку та може йти тільки вперед.

Кожна колонка таблиці буде оформлена в окремий нумерований список. Після того, як дані записано у ResultSet у вигляді таблиці, відповідний список буде отримувати інформацію, яка повинна бути у даній колонці. Для безперервного додання інформації у списки буде створено цикл, який працюватиме доки усі дані з ResultSet не буде додано до усіх списків.

```
List<String> driver = new ArrayList<>();
ResultSet rsTemp;
rsTemp = statementTmp.executeQuery(query);
```

while (rsTemp.next()) driver.add(rsTemp.getString("forename") + " " + rsTemp.getString("surname"));

Таблиці бібліотеки JavaFX можуть коректно працювати лише з деякими видами представлень даних. Одним з найкращих представлень є ObservableList.

ObservableList - це список, який має підтримку слухача, що дозволяє відслідковувати зміни коли вони відбулись. Такий список може набувати дуже великих розмірів. В нашому випадку ObservableList являє собою таблицю, яка буде складатись з декількох звичайних нумерованих списків, описаних вище. Після того, як ObservableList буде сформований остаточно, його дані буде перенесено та відображено у таблиці.[8]

Для оголошення списку ObservableList необхідно спочатку створити клас-модель. Такий клас складається з декількох складових частин:

- змінні - декілька змінних певного типу, які визначають скільки списків включатиме ObservableList, а в подальшому скільки колонок матиме таблиця. Тип змінної визначає елементи якого типу можуть знаходитись у цій колонці. Елементи інших типів необхідно приводити(конвертувати) до цього типу.
- конструктор - метод, названий іменем класу. Під час додавання списків до ObservableList, буде створено екземпляр класу через цей метод.
- Getters and Setters - методи для отримання та збереження даних.

Такого роду класи і стануть типом спостережуваних списків. Кількість класів залежить від кількості списків, а кількість списків - від кількості видів таблиць.

Можливість додавати слухачі до списків зробить застосунок динамічним. Зазвичай для оновлення даних потрібно використовувати додаткові елементи керування, що погіршує взаємодію користувача з додатком. В даному випадку, елементи інтерфейсу будуть змінюватись при зміні даних у спостережуваному списку.

public ObservableList<DriverChampionships> driverChampionships = FXCollections.observableArrayList();

for (int i = 0; i < position.size(); i++)
driverChampionships.add(new DriverChampionships(position.get(i), name.get(i),
nationality.get(i), "constructor.get(i)", points.get(i)));

Завдання та загальна концепція застосунку передбачає наявність декількох вікон інтерфейсу. Ці вікна між собою не поєднані і кожне з них має свій власний клас-контролер. Для того, щоб відобразити дані в одному вікні, потрібно отримати інформацію з іншого вікна про те, які дані користувач хоче побачити. Отже, до модуля вилучення даних потрібно додати механізм обміну даними між вікнами.

Такий інструмент в Java можна створити на основі класу, що реалізує шаблон проектування Singleton.

Шаблони проектування поділяються на декілька видів. Твірними називають ті, які абстрагують процес побудови об'єктів. Найпростішим з них є Singleton. Цей шаблон, представлений у вигляді класу, гарантує існування тільки одного об'єкту певного класу, а також дає змогу отримати до нього доступ з будь-якої ділянки коду.

Для деяких класів важливо, щоб існував тільки один екземпляр. Наприклад, хоча у системі може існувати декілька принтерів, може бути тільки один спулер. Повинна бути тільки одна файлова система та тільки один активний віконний менеджер.

Глобальна змінна не вирішує такої проблеми, бо не забороняє створити інші екземпляри класу.

Рішення полягає в тому, щоб сам клас контролював свою «унікальність», забороняючи створення нових екземплярів, та сам забезпечував єдину точку доступу. Це є призначенням шаблону Singleton.[9]

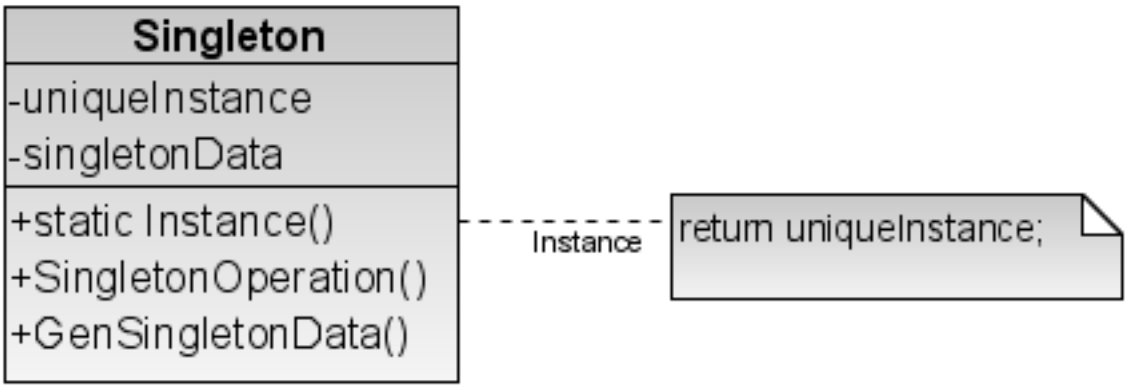


Рис. 2.4 Діаграма класів, що описує структуру шаблону Singleton

Так виглядає класична реалізація цього шаблону:

```
public class DataExtracter {
    private static final DataExtracter instance = new DataExtracter();
    public static DataExtracter getInstance(){
        return instance;
    }
}
```

У класі ми маємо приватну статистичну змінну instance для зберігання єдиного екземпляра нашого же класу. Приватний конструктор означає, що тільки сам клас може створювати свої екземпляри. Ми не можемо створити клас за допомогою оператора new. Для цього в класі є статичний публічний метод getInstance(), який і повертає екземпляр класу. До цього методу всі і будуть звертатись для створення екземпляра класу.

Після визначення елементів керування інтерфейсу, які і будуть містити дані для передачі, в цьому класі будуть створені необхідні змінні, в які будуть збережені елементи керування, та методи запису та отримання цих змінних(Getter and Setter).

Висновки до розділу 2

Для розв'язання поставленої задачі було написано застосунок мовою Java, через який користувач отримує необхідну йому інформацію. Налаштований локальний сервер забезпечує безперебійний доступ до бази даних.

За допомогою бібліотеки JavaFX спроектовано графічний інтерфейс, особливостями якого є зрозумілість, простота та швидкість. Він відповідає усім вимогам та дозволяє зручно підійти до аналізу даних предметної області.

Розроблено модуль вилучення інформації з бази даних. Він отримує необхідні дані згідно з сформованим запитом. Також створено декілька допоміжних класів для коректної роботи основного модуля.

РОЗДІЛ 3
ОСОБЛИВОСТІ РОБОТИ ІЗ ЗАСТОСУНКОМ

3.1. Рекомендації щодо використання

Для входу в інформаційну систему необхідно запустити метод main() класу Main. Після запуску застосунку, перед користувачем з'являється графічний інтерфейс. Першочергово необхідно звернути увагу на вибір параметрів запиту(рис. 3.1).

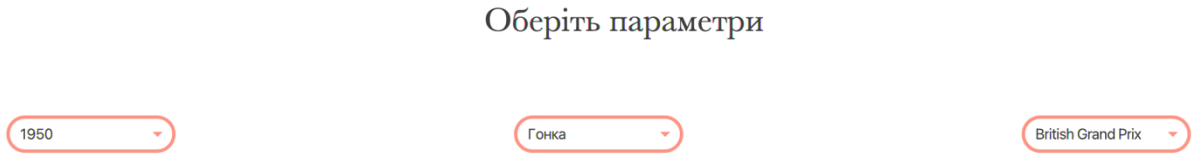


Рис. 3.1. Вибір параметрів запиту

До вибору пропонуються три основні параметри:

- рік - офіційно Формула-1 проходить, починаючи з 1950 року;
- тип даних - яку саме вибірку хоче отримати користувач;
- підтип - конкретна гілка в типовому дереві.

Рік представлено у вигляді чисел, які після завантаження головного вікна одразу записуються у нумерований список. Ці числа беруться з таблиці "seasons", де й зберігаються усі сезони та їх опис. Після завершення формування списку, його дані записуються в спостережуваний список для подальшого відображення в об'єкті класу ChoiceBox.

Список типових варіантів також визначений заздалегідь та формується на етапі завантаження інтерфейсу. Користувач може обрати з чотирьох варіантів:

- Гонка - етап чемпіонату світу Формула-1. При виборі цього параметру, останній список буде сформовано з тих етапів, що пройшли в році, обраному на першому кроці;
- Гонщик - пілот однієї з команд Формули-1. На останньому кроці до вибору буде запропоновано пілота, що приймав участь хоча б в одному етапі чемпіонату світу в році, обраному на першому кроці;
- Команда - один з конструкторів чемпіонату Формула-1. Третій випадаючий список міститиме назви команд-конструкторів, заявлених в чемпіонаті світу року, обраному раніше;
- Чемпіонат - світова першість в класі автоперегонів на болідах з відкритими колесами Формула-1. При виборі цього параметру, на останньому кроці користувач повинен обрати один з двох основних чемпіонатів в рамках Формули-1 - чемпіонат пілотів та чемпіонат команд;

Після того, як вибір зроблено, список з третім параметром одразу заповниться можливими варіантами, що відповідають першим двом умовам. Хоча не усі комбінації параметрів є можливими.

Для правильного проектування такої системи необхідно враховувати структурні особливості предметної області. Відомо, що до 1958 року чемпіонат команд не регламентувався та не проводився. Отже, не всі комбінації перших двох параметрів є справедливими. Розглянемо наступні рівняння:

$$(\text{Р} < 1958) + (\text{Т} = \text{"Команда"}) = \emptyset \quad (3.1)$$

або

$$(\text{Р} < 1958) + (T = \text{"Чемпіонат"}) + (\text{П} = \text{"Чемпіонат команд"}) = \emptyset \quad (3.2)$$

де Р - рік, Т - тип, П - підтип.

Такого роду комбінації можуть призвести до порожнього третього списку або взагалі до помилки в роботі застосунку. Для уникнення таких ситуацій до інтерфейсу користувача додано спеціальну мітку, яка повідомляє користувача про некоректність вибору параметрів.

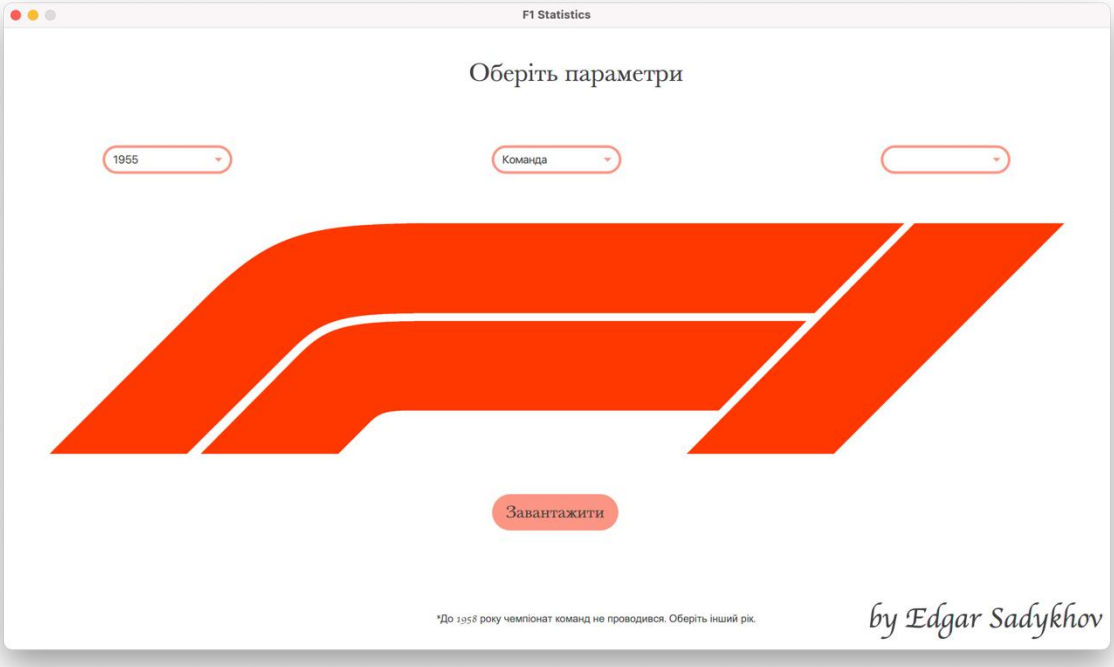


Рис. 3.2 Комбінація параметрів з рівняння (3.1)

Комбінація параметрів на рис. 3.2 показує, що список вибору підтипу порожній. В такому випадку користувач не може продовжити роботу із застосунком. Мітка в центральній нижній частині інтерфейсу сповіщає про це та пропонує обрати інший рік.

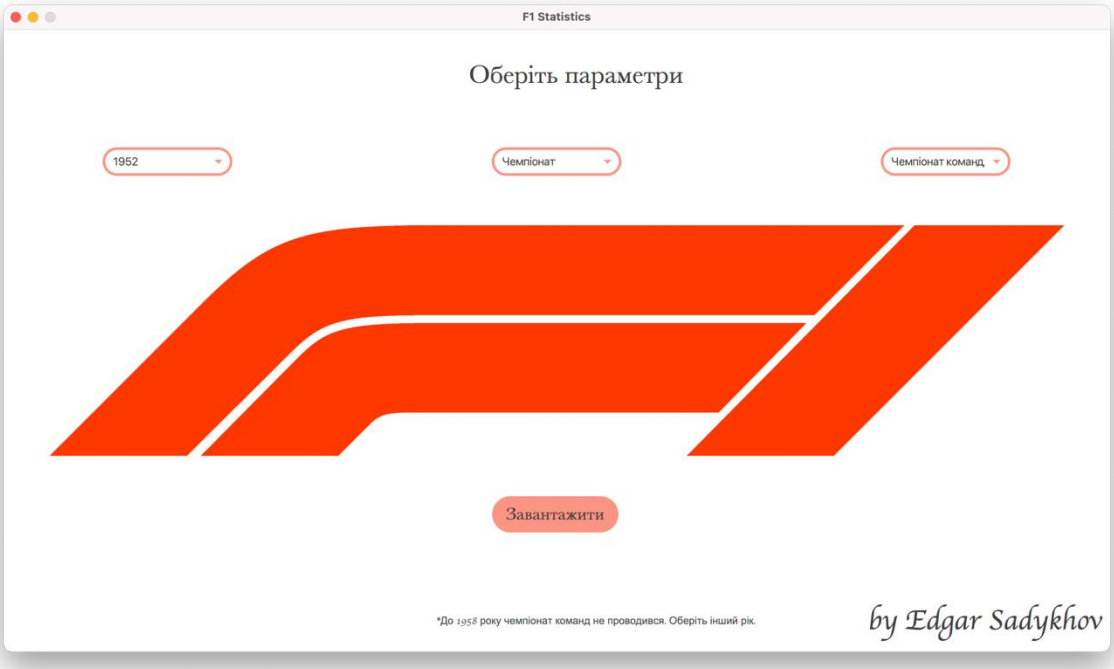


Рис. 3.3 Комбінація параметрів з рівняння (3.2)

Комбінація параметрів з рис. 3.3 також активує мітку. В цьому випадку застосунок не покаже жодних помилок, але таблиця з даними буде порожня. У цій ситуації рішення аналогічне - змінити рік.

Після того, як користувач закінчив підбір параметрів запиту, необхідно натиснути на кнопку "Завантажити". Ця дія відправить інформацію до модулю вилучення даних, змінить сцену у вікні та завантажить усі знайдені дані, що відповідають вказаним параметрам. Нова сцена має наступний вигляд:

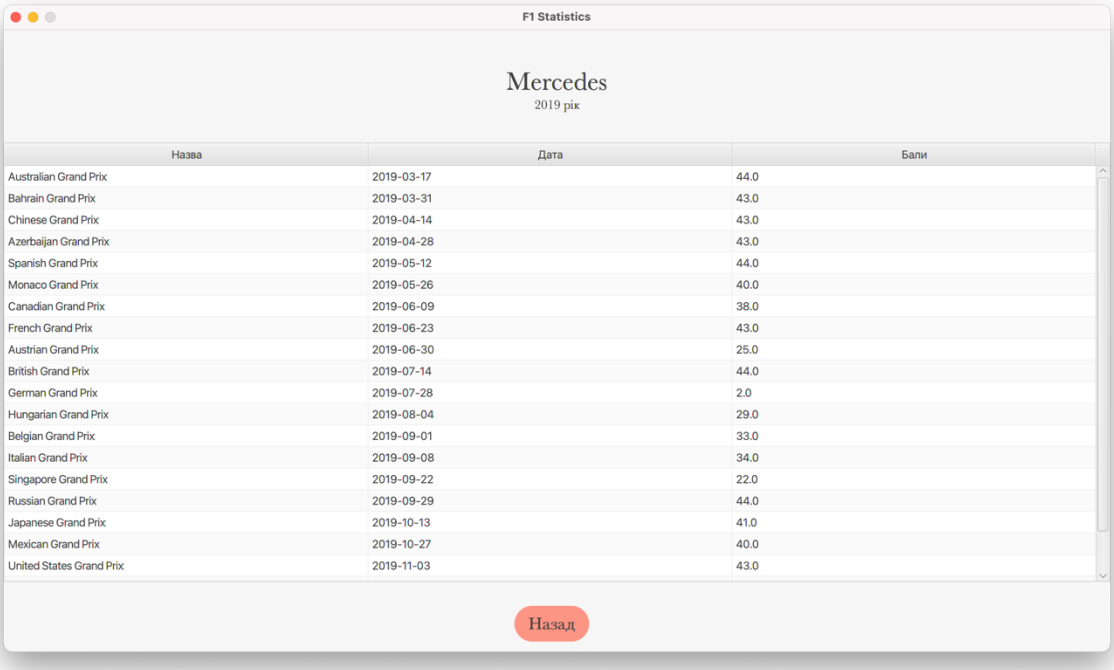


Рис. 3.4. Сцена інтерфейсу з табличним представленням даних

На рис. 3.4 видно, що дані, які отримані від запиту до серверу, структуровано в табличному представленні. Кількість стовбців, їх назва та тип даних залежать від обраних типу та підтипу під час визначення параметрів.

База даних має більше статистики, ніж відображається в таблиці. Вибірковість вилучення даних обумовлена тим, що деякі з них не є репрезентативними та відволікають увагу користувача під час аналізу. Кожна з таблиць містить стовбці, дані в більшості з яких впливають на кінцевий результат події, що розглядається в типовому дереві. Опис структури представлено в таблиці 3.1.

Таблиця 3.1.

Можливі варіації таблиці

Тип таблиці		Назва стовбців
Гонка		– Позиція на фініші; – Стартова позиція; – Ім'я пілота; – Назва команди; – Кількість кіл; – Найкращий час; – Кількість балів.
Пілот		– Назва гонки; – Дата; – Команда; – Позиція на фініші; – Кількість балів.
Команда		– Назва гонки; – Дата; – Кількість балів.
Чемпіонат	Чемпіонат пілотів	– Позиція; – Ім'я пілота; – Національність; – Назва команди; – Кількість балів.
	Чемпіонат команд	– Позиція; – Ім'я команди; – Національність; – Кількість балів; – Кількість перемог.

Об'єкт класу TableView має вбудовану функцію сортування. Після натискання на назву стовбця ,всю таблицю буде відсортовано в порядку "від меншого до більшого", спираючись на порядок саме цього стовбця. Якщо натиснути ще один раз, відбудеться аналогічне сортування, тільки тепер в порядку "від більшого до меншого" усієї таблиці.

3.2. Безпека та захищеність даних

В питаннях безпеки даних СКБД MySQL - один зі світових лідерів. Тут потрібно врахувати велику кількість можливих рішень, а також як вони впливають на безпеку MySQL-серверів та подібних додатків.

Ця система має підтримку встановлення складних паролів, обмеження доступу неавторизованих користувачів, захист бази даних від редагування або видалення та багатьох інших інструментів. Усі файли з даними, файли-журнали та файли інсталяції захищені та не можуть бути прочитані або перезаписані небажаними особами.[10]

Сервер MAMP дає можливість змінити порт для передачі даних. Таким чином зловмисники не отримають бажану інформацію, проводячи атаки на стандартні порти. Найуразливіші точки доступу можна закрити через брандмауер.

Мова програмування Java володіє інструментом, який називають системою управління пам'яттю на основі збору сміття. Такий підхід дозволяє видаляти тимчасові дані з оперативної пам'яті, які потенційно можуть стати уразливою точкою системи безпеки.[11]

Інтерфейс застосунку не дає можливість змінювати дані в базі або навіть в межах таблиці. Це дозволить уникнути випадкової зміни інформації, яка є достовірною та зібраною на основі фактів. Для додання нових сезонів та етапів чемпіонату світу необхідно користуватись застосунками для моделювання та проектування баз даних.

3.3. Тестування застосунку

Після закінчення написання програмного коду, було проведено тестування. Платформою для проведення тестів виступила бібліотека JUnit 5. Це фреймворк автоматичного тестування окремих ділянок застосунку, найчастіше методів або класів. В даному випадку об'єкт тестування - класи-моделі, які використовуються для зберігання інформації певних типів. Першочергово створюється клас, в якому і будуть зберігатись методи, написані для тестування моделей. Наприклад, так виглядає один з методів тестування класу RaceResults:

```
class RacesResultsTest {
    private RacesResults rr;
    @BeforeEach public void setUp() {
        rr = new RacesResults(null, "1", "Lorem Ipsum", "Mercedes", "50", "1:23.450", "25");
    }
    @Test void getFinishNull() {
        String nullFinish = null;
        assertEquals(rr.getFinish(), nullFinish);
    }
```

Загалом створено та запущено близько 50 автоматичних тестів. Більшість методів було написано за звичайною методикою, а декілька - для отримання негативних результатів. Усі тести пройшли успішно, що свідчить про правильну роботу застосунку.

Також було проведено декілька тестових сесій через графічний інтерфейс. Основною метою такого роду випробувань стала імітація реальних сценаріїв

використання та пошук невідповідностей елементів інтерфейсу з початковою концепцією.

Протягом тестувань не виявлено жодних проблем. Елементи інтерфейсу симетрично розташовані один відносно одного. Під час роботи застосунку не було знайдено критичних помилок, консоль та вкладка журналу з проблемами не відображали жодних повідомлень. Завершення роботи застосунку також проходить коректно. Усі служби та сервіси закінчують функціонування в "тихому" режимі.

Висновки до розділу 3

Отже, у даному розділі розгорнуто описано особливості використання застосунку. Так, представлено детальну інструкцію, яка крок-за-кроком розповідає користувачу його можливості.

Існує велика кількість комбінацій параметрів, які користувач може обрати для отримання необхідних даних. Деякі комбінації не є можливими через характерні особливості предметної області. Такі випадки враховано та регламентовано.

Створений застосунок має достатній рівень безпеки для персонального користування. Для його досягнення використано декілька вбудованих інструментів із функціоналу MySQL та Java.

Проведено ряд тестових випробувань застосунку. Після їх завершення можна стверджувати, що програмний комплекс працює коректно, без критичних помилок.

ВИСНОВКИ

В ході виконання дипломного проекту було розроблено систему відображення даних Формули-1 "F1 Statistics", яка необхідна профільним виданням, Інтернет-ресурсам та вболівальникам чемпіонату для аналізу минулих подій.

Програма написана мовою Java, а отже може бути запущена на будь-яких платформах та ОС, де підтримується JRE. Інтерфейс застосунку створено за допомогою ПЗ SceneBuilder, який дозволяє швидко проектувати графічні інтерфейси на основі бібліотеки JavaFX.

Для організації даних залучено реляційну СКБД MySQL, з якою і взаємодіє застосунок. В цій базі даних існує декілька таблиць, в яких знаходяться статистичні дані предметної області. Цю інформаційну структуру розгорнуто на локальному сервері, відкритому через MAMP.

Створений застосунок дає можливість отримати табличне представлення необхідної інформації, відсортувати її та підготувати до аналізу. Такий підхід значно полегшує набути знання з предметної області, уникнути помилкових даних та непідтверджених джерел.

Модернізувати програмний комплекс можна шляхом додання нових елементів інтерфейсу, підключення нових програмних модулів, розширення основного та імплементація додаткового функціоналу, розміщення застосунку в якості Java-аплету на HTML-сторінці в мережі Інтернет.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Формула-1 – Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Формула-1 (дата звернення: 10.04.2021р.). – Назва з екрану
2. Фаулер М. Архитектура корпоративных программных приложений / Мартин Фаулер. – Москва: Вильямс, 2007. – 544 с.
3. Зручність використання програмного забезпечення [Електронний ресурс] – Режим доступу: https://studopedia.org/8-131357.html (дата звернення: 21.05.2021р.). – Назва з екрану
4. MAMP - ваша местная среда разработки [Електронний ресурс] – Режим доступу: mamp.info/ru/mamp/mac/ (дата звернення: 11.05.2021р.). – Назва з екрану
5. phpMyAdmin [Електронний ресурс] – Режим доступу: https://www.phpmyadmin.net/ (дата звернення: 22.04.2021р.). – Назва з екрану
6. Что такое XML [Електронний ресурс] – Режим доступу: https://msiter.ru/tutorials/uchebnik-xml-dlya-nachinayushchih/chto-takoe-xml (дата звернення: 17.05.2021р.). – Назва з екрану
7. Одинак (шаблон проєктування) – Вікіпедія [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Одинак_(шаблон_проєктування) (дата звернення: 14.05.2021р.). – Назва з екрану
8. Маншин Т. JavaFX 2.0. Разработка RIA-приложений / Тимур Маншин. – Санкт-Петербург: БХВ-Петербург, 2012. – 320 с.
9. ChoiceBox(JavaFX 8) [Електронний ресурс] – Режим доступу: https://docs.oracle.com/ (дата звернення: 17.05.2021р.). – Назва з екрану
10. MySQL :: Security in MySQL :: 1 Security [Електронний ресурс] – Режим доступу: https://dev.mysql.com/doc/mysql-security-excerpt/5.7/en/security.html дата звернення: 21.05.2021р.). – Назва з екрану

11. The Basics of Java Security | Baeldung [Електронний ресурс] – Режим доступу: https://www.baeldung.com/java-security-overview.html (дата звернення: 21.05.2021р.). – Назва з екрану